PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Igor Wanderley Cavalcanti

ANÁLISE DOS DADOS DO TMDB E MOVIELENS E ALGUNS MODELOS PREDITIVOS DE GÊNEROS DE FILMES

Belo Horizonte
2020

Igor Wanderley Cavalcanti

Análise dos dados do TMDB e movielens e alguns modelos preditivos de gêneros de filmes

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte
2020

SUMÁRIO

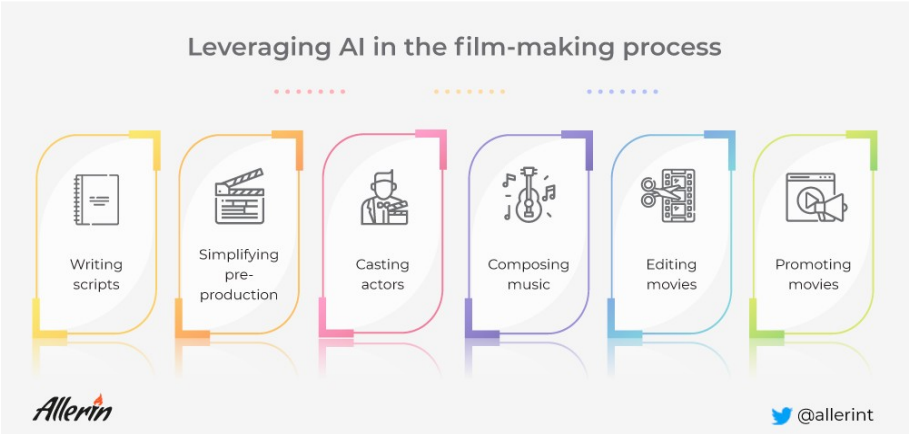
1. Introdução

1.1. Contextualização

A Inteligência Artifcial, Machine Learning, Data Science estão cada vez mais presentes em nossas vidas. Desde chatbots de atendimento a sistemas de recomendação de filmes que nos indica qual deveria ser o próximo filme que deveríamos assistir.

Na indústria de filmes a Data Science e Machine Learning se fazem mais presentes a cada dia. Estão em uso sistemas que conseguem prever o sucesso ou fracasso de um filme, quais cores devem ser utilizadas no marketing dos filmes ou que atores garantiriam o sucesso do filme. A 20th Century Fox usou servidores do Google e utilizando o TensorFlow criou o Merlin, um sistema experimental de predição de audiência e de recomendação de filmes. Nos testes, a ferramenta analisou o trailer do filme 'Logan' para prever que outros filmes seriam interessantes para os que assistiram ao filme. Entre os indicados: X-Men: Apocalipse, Dr. Estranho, Batman vs Superman e Esquadrão Suicida. Para os serviços de streaming (Netflix, Amazon Prime, Disney+, etc), serviços de recomendação são essenciais para retenção de usuários.

A Inteligência Artificial e o Data Science estão cada vez mais presentes em nossa vida, mesmo sem perceber sua presença. Segue abaixo uma figura do Allerin onde mostra onde a Inteligência Artificial está entrando na produção de filmes



Inteligência Artificial na industria de filmes. fonte: Allerin

1.2. O problema proposto

O trabalho consistirá em fazer uma análise exploratória dos dados tentando obter alguns insights sobre os conjuntos de dados que veremos a seguir. Faremos uma breve análise estatística dos datasets e, por fim, gerar alguns modelos de predição de gênero do filme a partir do overview do filme. Nessa parte de machine learning iremos testar alguns algoritmos e algumas das técnicas de tratamento de datasets com classes desbalanceadas. Na preparação dos dados para serem utilizados nos algoritmos de aprendizagem de máquina poderemos ver que alguns gêneros tem muito mais ocorrências que os demais.

Como vimos anteriormente, temos diversas possibilidades de aplicação da Inteligência Artificial/Data Science na indústria de filmes. O que mais conhecemos e utilizamos são os sistemas de recomendação presentes nos serviços de streaming como a Netflix, Amazon Prime, HBO Go, etc. Esses serviços utilizam sistemas de recomendação para que seus usuários possam ter um melhor aproveitamento do conteúdo disponibilizado e tentar garantir sua retenção.

Na internet podemos encontrar diversos exemplos de sistemas de recomendação em cima de datasets da movielens (https://grouplens.org/datasets/movielens/), do 'The Movies Database' - TMDB (https://www.themoviedb.org/), do Internet Movie Database – IMDb (https://www.imdb.com/interfaces/), entre vários datasets customizados que podem ser encontrados no Kaggle (https://www.kaggle.com/). Temos exemplos de machine learning utilizando diversos métodos, tais como: Regressão Linear, Filtro Colaborativo, Agrupamentos, Decomposição de Matrizes e Deep Learning. Ao tentar resolver um problema de classificação de gêneros de filmes a partir de seus overviews, temos uma maneira mais precisa de avaliar a eficiência dos algoritmos, ao contrário do que encontramos ao tentar resolver problemas de sistemas de recomendação. Ainda podemos encontrar vários exemplos de análise de sentimentos de comentários sobre filmes.

Neste trabalho iremos construir modelos de predição de gêneros de filmes a partir do campo overview associado ao filme. Mas por que fazer um sistema de predição de gêneros de um filme? Para a geração dos modelos de predição de gêneros trabalharemos com a junção de dados de alguns datasets, trataremos o

básico de processamento de linguagem natural, trabalharemos com dados onde as classes alvo estão desbalanceadas, além de trabalhar com um módulo otimizador de modelos do scikit-learn (sklearn). Na internet, os poucos exemplos que encontramos com problemas parecidos com esse, utilizam sinopse, ou seja, um texto maior que o overview do filme, dando mais insumos para que a aplicação consiga prever o gênero do filme com mais precisão. Dos exemplos que encontramos na internet, a maioria deles não passa de 0.5 no score de avaliação (dependendo do script pode ser precision_score, f1_score ou accuracy_score). Em alguns trabalhos onde o score de avaliação passava de 0.8, olhando melhor o código, percebe-se que a medida utilizada foi a 'micro' e não a 'macro'. A medida 'macro' é uma média do score para todas as classes, enquanto que a 'micro' fornece a maior acurácia encontrada em alguma classe. Um dos desafios que pretendemos vencer é ter um score (macro) maior que 0.5.

No trabalho teremos os seguintes notebooks python:

- AnaliseExploratoriaMovieLensTMDB: Aqui fazemos uma análise e tratamento inicial dos datasets. Esse notebook traz uma análise gráfica dos dados dos datasets onde poderemos tirar alguns insigths e responder algumas perguntas sobre os dados
- AnaliseEstatisticaBasicaMovieLensTMDB: Aqui fazemos tratamento necessário para fazer uma análise estatística básica em cima das notas e médias de notas atribuídas aos filmes.
- DadosTMDB: Esse script é responsável por trazer os overviews dos filmes do dataset do movielens utilizando a API do TMDB.
- Notebooks de Machine Learning: Temos 5 notebooks, um para cada algoritmo de machine learning utilizado. Os notebooks são praticamente iguais, o que muda são os parâmetros passados ao pipeline de crossvalidation e os algoritmos em si de aprendizagem de máquina.

2. Coleta de Dados

A origem dos datasets utilizados:

The movie Database - TMDB:

https://www.kaggle.com/tmdb/tmdb-movie-metadata

Desse conjunto trabalharemos com o arquivo tmdb_5000_movies.csv. Esse dataset foi feito utilizando a API provida pelo TMDB. Sua última atualização foi 3 anos atrás.

MovieLens (https://grouplens.org/datasets/movielens/25m)

O Group Lens coletou e disponibilizou dados a partir do site do MovieLens (http://movielens.org). Neste trabalho utilizaremos os datasets disponibilizados pelo movielens. O dataset utilizado foi disponibilizado em dezembro/2019.

Por que utilizar os dois datasets? O dataset do movielens tem o identificador do filme no TMDB. O TMDB traz informações de overview e outras informações interessantes sobre os filmes. Algumas vantagens do movielens é: a lista bem definida dos gêneros dos filmes, termos um dataset de 62423 registros de filmes que depois de realizar a extração de dados utilizando a API do themoviedb.org e tratar os dados teremos um dataset de 56081 registros com a coluna overview disponível, que será muito útil em nossos modelos de machine learning. Os datasets foram baixados em agosto/2020. Já a extração dos dados, foram feitas no fim de setembro/2020. O script 'DadosTMDB' de extração de dados encontra-se no apêndice deste trabalho e no github

2.1. Descrição dos dados utilizados

movielens		
ratings.csv		
Coluna	**Descrição**	**Tipo**
userId	Identificação do usuário que atribuiu a nota	Inteiro
movieId	Identificação do filme que recebeu a nota	Inteiro
rating	Nota recebida pelo filme	ponto flutuante
timestamp	Data em que o filme recebeu a nota	inteiro

movielens		
movies.csv		
Coluna	**Descrição**	**Tipo**
movieId	Identificador do filme	Inteiro
title	Título do Filme	texto
genres	Gêneros do filme em csv separados por '|'	texto

movielens		
tags.csv		
Coluna	**Descrição**	**Tipo**
userId	Identificação do usuário que atribuiu a tag ao filme	inteiro
movieId	Identificador do filme	inteiro
tag	Título do Filme	texto
timestamp	Gêneros do filme	inteiro

movielens		
links.csv		
Coluna	**Descrição**	**Tipo**
userId	Identificação do usuário que atribuiu a tag ao filme	inteiro
movieId	Identificador do filme	inteiro
imdbId	Identificador do filme no imdb	inteiro
tmdbId	Identificador do filme no tmdb	inteiro

TMDB		
tmdb_5000_movies.csv		
Coluna	**Descrição**	**Tipo**
budget	Valor de orçamento do filme	inteiro
genres	Gêneros de filme em que o filme pode ser encaixado	json
homepage	Homepage oficial do filme	texto
id	Identificação do filme	inteiro
keywords	Palavras chaves associadas ao filme	json
original_language	Lingua original em que o filme foi gravado	texto
original_title	Titulo original do filme	texto
overview	Sinopse do Filme	texto
popularity	Popularidade do filme calculada pelo próprio TMDB	ponto flutuante
production_companies	Companhias que fizeram parte da produção do filme	json
production_countries	Países em que o filme foi filmado	json
release_date	Data de Lançamento	texto
revenue	Retorno Financeiro	inteiro
runtime	Tempo de duração do filme em minutos	ponto flutuante
spoken_languages	Línguas faladas no filme	json
status	Se o filme já foi lançado ou não	texto
tagline	Subtítulo do Filme ou chamada rápida do filme (propaganda)	texto
title	Título do Filme	texto
vote_average	Nota média do filme	ponto flutuante
vote_count	Quantidade de votos que o filme recebeu	inteiro

3. Processamento/Tratamento de Dados

3.1. TMDB

- Registros: 4803
- Colunas: 20

Inicialmente analisaremos informações estatísticas e fazer limpeza dos dados, quando necessário, onde tiraremos informações ‘estranhas’ dos datasets. Isso somente para a análise e exploração dos dados, pois para o modelo de machine learning será feito um tratamento específico descrito posteriormente, além de que as discrepâncias encontradas nesta seção são irrelevantes para os modelos de aprendizado de máquina.

Vamos começar nossa análise pelas notas atribuídas aos filmes e suas médias. Para fazermos análise dos dados das notas e suas médias, seja de todos os filmes, seja de um filme específico, qual a confiança de uma média de um filmes que tem 5 avaliações e um filme que tem 1000. Podemos comparar essas notas?

3.1.2. Dados estatísticos básicos

Segue a saída do describe do tmdb_movies_dataset.csv:

	count	mean	std	min	25%	50%	75%	max
budget	4803.0	29045039,88	40722391,26	0.0	790000,00	15000000,00	40000000,00	380000000,00
id	4803.0	57165,48	88694,61	5.0	9014,50	14629,00	58610,50	459488,00
popularity	4803.0	21,49	31,82	0.0	4,67	12,92	28,31	875,58
revenue	4803.0	82260638,65	162857100,94	0.0	0,00	19170001,00	92917187,00	2787965087,00
runtime	4801.0	106,88	22,61	0.0	94,00	103,00	118,00	338,00
vote_average	4803.0	6,09	1,19	0.0	5,60	6,20	6,80	10,00
vote_count	4803.0	690,22	1234,59	0.0	54,00	235,00	737,00	13752,00

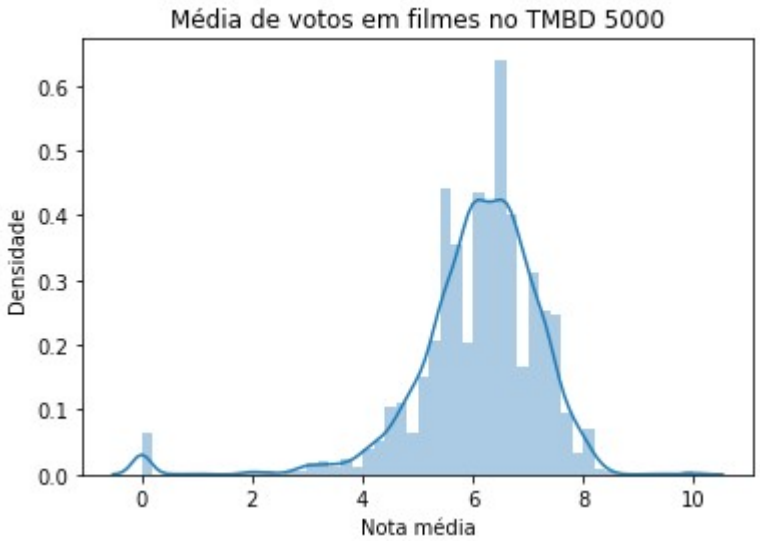
Na tabela fornecida pelo describe, podemos ver algumas informações estatísticas básicas das colunas numéricas do dataset. Na tabela podemos verificar a mediana (que figura na coluna 50%, e que representa o valor que divide o conjunto de dados ao meio) e os quartis (25% e 75%). Podemos ver, por exemplo, que o valor mínimo para "vote_average” é 0, e o máximo é 10. Dos dados mostrados acima vimos que a contagem de linhas de runtime está diferente das outras colunas numéricas, vamos verificar que colunas possuem valores nulos?

```
tmdb.isnull().sum()
```

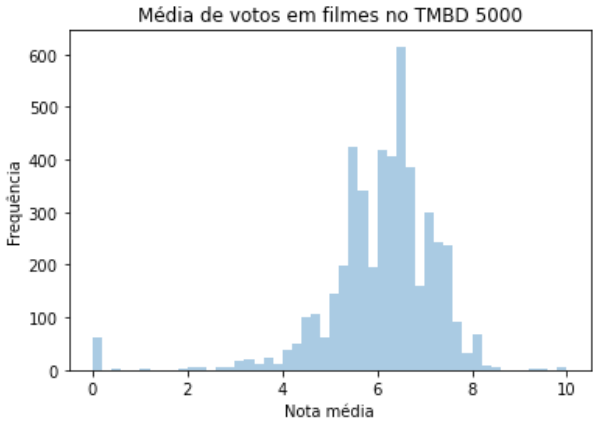
```
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0
dtype: int64
```

Pela figura acima podemos ver que algumas colunas possuem valores nulos. Devemos excluir as linhas que contém esses valores? Por enquanto não, pois iremos fazer análises dos dados em cima das colunas vote_average e vote_count.

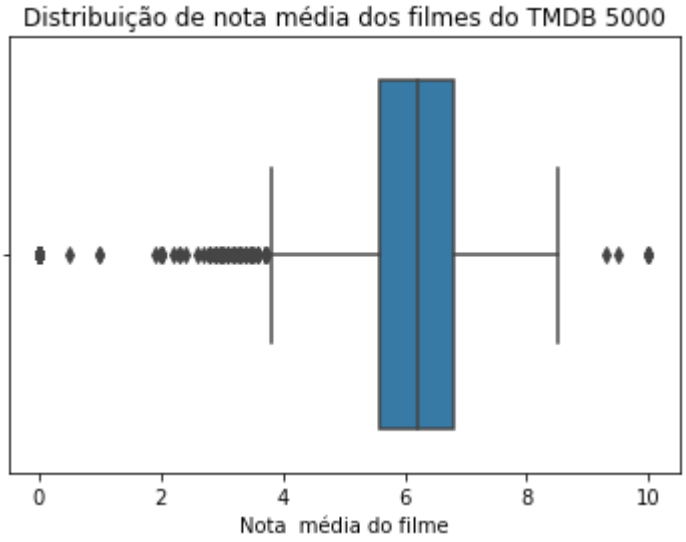
3.1.3. Verificação da nota média graficamente



No gráfico acima podemos ver que existem algumas notas zero, mas não sabemos quantas. Podemos ter uma ideia de quantidade se transformarmos o gráfico acima em um gráfico de frequências



Vamos ver como fica o boxplot do dataset. No boxplot poderemos ver a mediana e os quartis das notas



Uma informação importante que podemos extrair dos gráficos acima é que existem uma quantidade razoável de filmes com médias 0 ou 10. Seria incomum isso, pois como o campo é uma média, qualquer nota diferente de 0 ou 10 mudaria essas médias. Isso pode indicar que alguns filmes estão sem notas, ficando com média 0 e também que alguns filmes devem ter recebido pouquíssimas notas.

Vamos verificar notas 0:

```
tmdb.query('vote_average == 0 ')[['title','vote_average','vote_count']].head(5)
```

	title	vote_average	vote_count
1464	Black Water Transit	0.0	0
3669	Should've Been Romeo	0.0	0
3670	Running Forever	0.0	0
3852	The Secret	0.0	0
3855	Time to Choose	0.0	0

```
tmdb.query('vote_average == 0 ').shape
```

```
(63, 20)
```

```
tmdb.query('vote_average == 0 and vote_count >0 ').shape
```

```
(1, 20)
```

```
tmdb.query('vote_average == 0 and vote_count >0 ')[['title','vote_average','vote
```

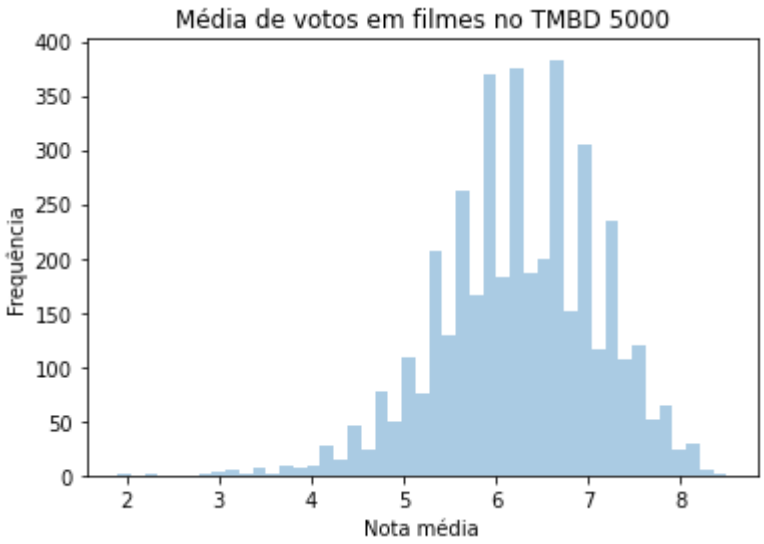
	title	vote_average	vote_count
4400	Sparkler	0.0	1

Com os dados acima, podemos verificar que outras colunas têm a sua importância, a coluna 'vote_count'. A maioria dos filmes com média 0 não possuem votos. Essas notas nos dizem que os filmes seriam ruins, o que não é necessariamente verdade. Vamos remover esses filmes do dataframe. Ainda há o caso de um filme onde recebeu uma única nota 0. Esse filme também será retirado do dataframe por não ser significativo para conclusões estatísticas. Iremos manter no dataframe os registros onde o mínimo de notas seja 20. Em seguida veremos as características do novo conjunto de dados:

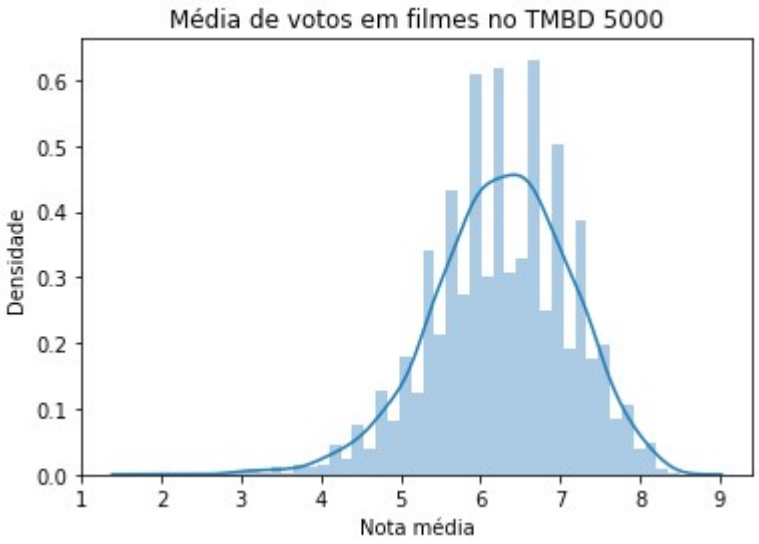
	count	mean	std	min	25%	50%	75%	max
budget	4138	33339515,12	42240392,37	0,00	4000000,00	20000000,00	45000000,00	380000000,00
id	4138	47549,11	78793,78	5,00	7942,50	12086,50	44935,25	417859,00
popularity	4138	24,75	33,14	0,01	7,63	16,17	31,79	875,58
revenue	4138	95346486,05	171892779,14	0,00	1013447,00	31905299,00	110224303,00	2787965087,00
runtime	4138	109,04	20,74	0,00	95,00	105,00	119,00	338,00
vote_average	4138	6,26	0,86	1,90	5,70	6,30	6,90	8,50
vote_count	4138	799,88	1297,04	20,00	108,00	325,00	882,00	13752,00

A nota média mínima passou a ser 1.90. Isso significa que todos os filmes tiveram pelo menos uma nota diferente de 0. Perceba que o máximo do campo vote_average também sofreu altereções. Passou a ser 8.5. Isso quer dizer que os filmes no nosso conjunto que apresentavam uma média 10 realmente possuíam pouquíssimos votos.

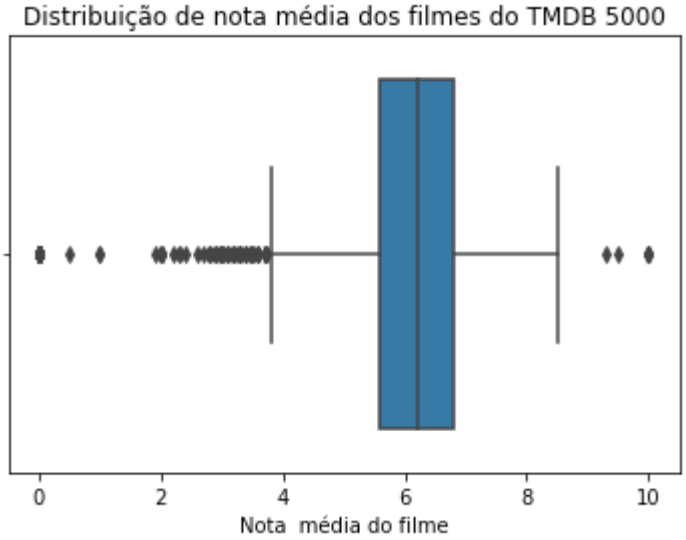
Vamos ver como ficaram os gráficos depois da exclusão dos filmes:



O histograma não apresenta mais registros com notas 0 ou 10. Vejamos agora a densidade de notas



Temos uma curva semelhante a uma distribuição normal, mas que não é exatamente simétrica, já que possui uma densidade um pouco maior na parte esquerda do gráfico. Veja, também, que analisando essa divisão a partir de uma média visual, existe uma amplitude de valores possíveis maior à esquerda (de 0 até 6) do que na direita (de 6 até 10).



Com o boxplot do novo conjunto de dados podemos ver que o conjunto vote_average é parecido com uma distribuição normal, apesar de ter uma densidade maior abaixo da mediana

Nós analisamos visualmente a média dos filmes do TMDB 5000, e percebemos que seu comportamento é parecido com uma distribuição normal, com exceção do lado esquerdo do gráfico.

Agora, vamos comparar esse conjunto com os dados do movielens, procurando saber se esse comportamento se repete.

3.2. MovieLens

Agora, vamos comparar esse conjunto do TMDB com os dados do movielens, procurando saber se esse comportamento se repete.

Para o movielens vamos carregar alguns datasets, o primeiro será o ratings.csv, esse dataset contém as notas atribuídas por um usuário a um filme.

- Registros: 25.000.095 linhas
- Colunas: 4

3.2.1. Dados estatísticos básicos

	count	mean	std	min	25%	50%	75%	max
userId	25000095,00	81189,28	46791,72	1,00	40510,00	80914,00	121557,00	162541,00
movieId	25000095,00	21387,98	39198,86	1,00	1196,00	2947,00	8623,00	209171,00
rating	25000095,00	3,53	1,06	0,50	3,00	3,50	4,00	5,00
timestamp	25000095,00	1215601443,12	226875808,06	789652009,00	1011747245,00	1198868375,00	1447205341,50	1574327703,00

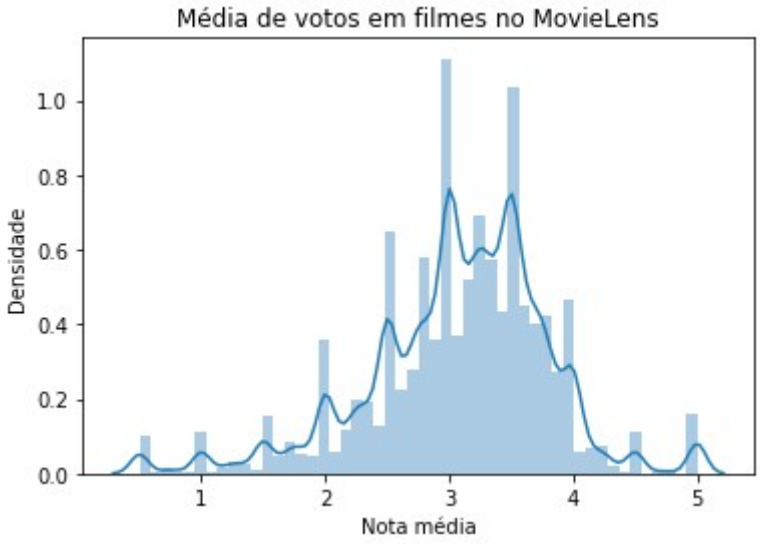
No movielens as notas variam de 0 a 5, já podemos perceber que o valor mínimo da nota não é zero.

3.2.2. Tratamento Inicial de Dados

No dataset arquivo rankings.csv não temos a nota média para cada filme.

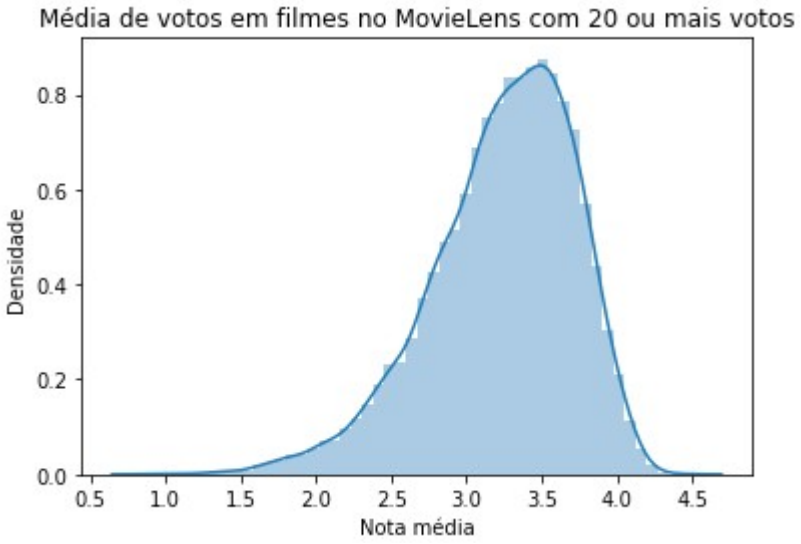
```
average_vote = notas.groupby("movieId").mean()["rating"]
average_vote.head()
```

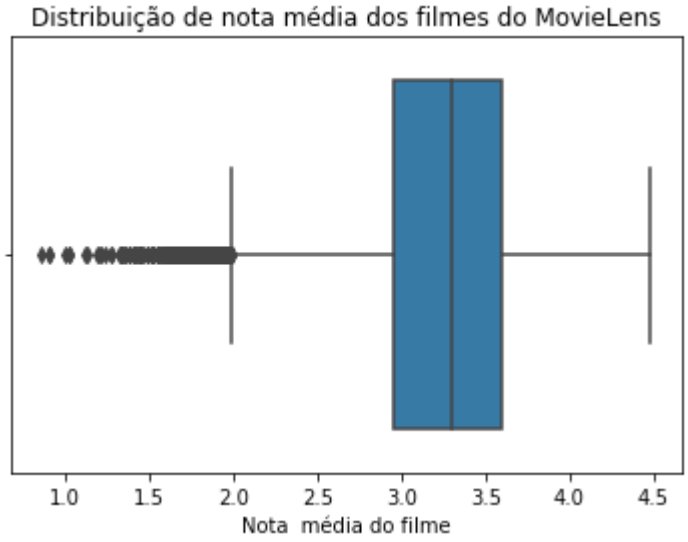
Vamos ver o gráfico das médias:



Podemos perceber que o gráfico não mostra filmes com média 0. Mas podemos ver que existem filmes com médias 5, assim podemos esperar que teremos filmes com poucos votos

Vamos ver um gráfico de distribuição (densidade) das médias dos votos de filmes com 20 ou mais votos e o boxplot:





Podemos ver que os quartis à esquerda são mais densos que os da direita.

Esse seria o tratamento e processamento de dados básico. Na parte de machine learning será feito seu próprio tratamento de dados para adaptação dos dados aos modelos escolhidos

3.1.2. Tratamento de dados para a análise exploratória

No dataset do TMDB fiz um tratamento para que possamos consultar somente o ano em que o filme foi lançado, já que o dataset traz a data de lançamento no formato YYYY-MM-DD. Outro tratamento feito foi para que os gêneros ficassem no mesmo formato que os gêneros do movielens

```
from ast import literal_eval
tmdb['genres'] = tmdb['genres'].fillna('[]').apply(literal_eval).\
apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
tmdb['genres'] = ['|'.join(map(str, l)) for l in tmdb['genres']]
tmdb['year'] = pd.DatetimeIndex(tmdb['release_date']).year
tmdb[['genres','release_date','year']].head(3)
```

	genres	release_date	year
0	Action|Adventure|Fantasy|Science Fiction	2009-12-10	2009.0
1	Adventure|Fantasy|Action	2007-05-19	2007.0
2	Action|Adventure|Crime	2015-10-26	2015.0

Foi feito um tratamento nos gêneros tanto no movielens como no TMDB para que os gêneros virassem campos binários nos dois conjuntos, facilitando, assim, na contagem dos gêneros para cada filme.

```
generos = movies.genres.str.get_dummies()
generos_tmdbb = tmdb.genres.str.get_dummies()
generos.head()
```

	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror
0	0	0	1	1	1	1	0	0	0	1	0	0
1	0	0	1	0	1	0	0	0	0	1	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	1	0	0	1	0	0	0
4	0	0	0	0	0	1	0	0	0	0	0	0

No Movie Lens tivemos que fazer um tratamento para cálculo de votos e nota média por filme, pois o dataset movies.csv não vem com essa informação. Conseguimos obter essa informação com o dataset ratings.csv. Depois de calcularmos as informações, juntamos os dados no dataframe do Movie Lens

```
notas = pd.read_csv("../../../dados/ml-25m/ratings.csv")
notas.head()
```

	userId	movieId	rating	timestamp
0	1	296	5.0	1147880044
1	1	306	3.5	1147868817
2	1	307	5.0	1147868828
3	1	665	5.0	1147878820
4	1	899	3.5	1147868510

Obtendo o dataset de notas

```
average_vote = notas.groupby("movieId").mean()["rating"]
average_vote.head()
```

```
movieId
1    3.893708
2    3.251527
3    3.142028
4    2.853547
5    3.058434
Name: rating, dtype: float64
```

Calculando as médias por filme

```
movies = movies.join(average_vote, on='movieId')
```

```
count_vote = notas.groupby("movieId").count()
count_vote = count_vote.drop('rating', 1)
count_vote = count_vote.drop('timestamp', 1)
count_vote.rename(columns={'userId':'qtd_votos'})
count_vote.columns = ['qtd_votos']
count_vote.head(3)
```

	qtd_votos
movieId	
1	57309
2	24228
3	11804

```
movies = movies.join(count_vote, on='movieId')
movies.head(3)
```

	movieId	title	genres	rating	qtd_votos
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.893708	57309.0
1	2	Jumanji (1995)	Adventure|Children|Fantasy	3.251527	24228.0
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.142028	11804.0

Cálculo dos votos por filme e o resultado final do dataframe

3.1.2. Tratamento de dados para o machine learning

Para geração dos modelos de machine learning com processamento de linguagem natural tivemos que fazer tratamento nas colunas de gêneros e overview. A coluna genres vêm no formato json, a coluna genres tem que virar array. Vamos ver as colunas citadas em seu formato original:

```
1 tmdb[['genres', 'overview']]
```

	genres	overview
0	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	In the 22nd century, a paraplegic Marine is di...
1	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	Captain Barbossa, long believed to be dead, ha...
2	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	A cryptic message from Bond's past sends him o...
3	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	Following the death of District Attorney Harve...
4	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	John Carter is a war-weary, former military ca...
...	...	...
4798	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	El Mariachi just wants to play his guitar and ...
4799	[{"id": 35, "name": "Comedy"}, {"id": 10749, "...	A newlywed couple's honeymoon is upended by th...
4800	[{"id": 35, "name": "Comedy"}, {"id": 18, "nam...	"Signed, Sealed, Delivered" introduces a dedic...
4801	[]	When ambitious New York attorney Sam is sent t...
4802	[{"id": 99, "name": "Documentary"}]	Ever since the second grade when he first saw ...

Vamos começar transformando a coluna de gêneros arrays, o processo consiste em processar o json, retirando o valor da chave 'name' do json, agrupando-os em um array e, por fim, substituindo a coluna antiga com o novo valor.

Vejamos o processo de transformação descrito anteriormente:

```
1 tmdb['genres'] = tmdb['genres'].fillna('[]').apply(literal_eval)
2 .apply(lambda x: [i['name'] for i in x] if isinstance(x, list) else [])
3 tmdb.head(3)
```

	budget	genres	homepage	id	keywords	original_language	original_title
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar
1	300000000	[Adventure, Fantasy, Action]	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End
2	245000000	[Action, Adventure, Crime]	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre

Tranformando a lista JSON de gêneros em um array

A coluna ‘overview’ tem alguns valores nulos, vamos tratá-los:

```
tmdb.isnull().sum()
```

```
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0
dtype: int64
```

```
tmdb['overview'].fillna('', inplace=True)
tmdb.isnull().sum()
```

```
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
overview                   0
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0
dtype: int64
```

Para o machine learning, iremos utilizar somente um conjunto de dados do TMDB e movielens. Iremos utilizar somente as colunas ‘genres’, ‘overview’, ‘title’.

Para a nossa abordagem da criação dos modelos temos um problema em específico que será descrito com detalhes na seção 5 – Criação de Modelos de Machine Learning.

O que precisaremos fazer na coluna de gêneros:

1. Manter o array de gêneros, mas com somente um gênero no array;
2. Fazer um tratamento para equilibrar a quantidade de gêneros, no caso, nossas classes alvo, assim, faremos um balanceamento das quantidades de registros para cada classe

Para fazer a transformação do array, usaremos uma instrução mais complexa onde será criada uma linha para cada elemento do array, mantendo todos os dados mudando somente os gêneros:

```
title_df = tmdb[["overview","genres","title"]]
filmes_balanceados = title_df.copy(deep=True)
```

```
lst_col = 'genres'

filmes_balanceados2 = pd.DataFrame({
      col:np.repeat(filmes_balanceados[col].values, filmes_balanceados[lst_col].str.len())
      for col in filmes_balanceados.columns.drop(lst_col)}
    ).assign(**{
    lst_col:np.concatenate(filmes_balanceados[lst_col].values)})[filmes_balanceados.columns]
```

A instrução anterior coloca cada gênero como string na coluna, precisamos colocar a string de volta em um array, mas faremos isso depois de balancear os gêneros:

```
title_df.head(3)
```

	overview	genres	title
0	In the 22nd century, a paraplegic Marine is di...	[Action, Adventure, Fantasy, Science Fiction]	Avatar
1	Captain Barbossa, long believed to be dead, ha...	[Adventure, Fantasy, Action]	Pirates of the Caribbean: At World's End
2	A cryptic message from Bond's past sends him o...	[Action, Adventure, Crime]	Spectre

```
filmes_balanceados2.tail(3)
```

	overview	genres	title
12157	"Signed, Sealed, Delivered" introduces a dedic...	Romance	Signed, Sealed, Delivered
12158	"Signed, Sealed, Delivered" introduces a dedic...	TV Movie	Signed, Sealed, Delivered
12159	Ever since the second grade when he first saw ...	Documentary	My Date with Drew

Faremos agora o balanceamento dos gêneros, o método escolhido para uso de classes desbalanceadas foi o upsample, isso implica em aumentar as quantidades de registros de cadas gênero, assim, todos ficarão com o mesmo número de registros. Vejamos as quantidades antes:

```
filmes_balanceados2['genres'].value_counts()
```

```
Drama              2297
Comedy             1722
Thriller           1274
Action             1154
Romance             894
Adventure           790
Crime               696
Science Fiction     535
Horror              519
Family              513
Fantasy             424
Mystery             348
Animation           234
History             197
Music               185
War                 144
Documentary         110
Western              82
Foreign              34
TV Movie              8
Name: genres, dtype: int64
```

Faremos com que todos os gêneros tenham 2297 registros. Primeiro passo será separar cada gênero em um dataframe diferente:

Separando cada gênero em dataset diferente para poder fazer resample dos dados - Tratamento de classes desbalanceadas

```
df_Drama = filmes_balanceados2[filmes_balanceados2['genres']=='Drama']
df_Comedy  = filmes_balanceados2[filmes_balanceados2['genres']=='Comedy']
df_Thriller   = filmes_balanceados2[filmes_balanceados2['genres']=='Thriller']
df_Action  = filmes_balanceados2[filmes_balanceados2['genres']=='Action']
df_Romance   = filmes_balanceados2[filmes_balanceados2['genres']=='Romance']
df_Adventure = filmes_balanceados2[filmes_balanceados2['genres']=='Adventure']
```

Em seguida faremos um resample em cada novo dataframe para que todos fiquem com a mesma quantidade de registros do dataframe do gênero ‘Drama’

```
df_Comedy  = resample(df_Comedy,
                      replace=True,
                      n_samples=2297,
                      random_state=123)
df_Thriller   = resample(df_Thriller,
                      replace=True,
                      n_samples=2297,
                      random_state=123)
df_Action  = resample(df_Action,
                      replace=True,
                      n_samples=2297,
                      random_state=123)
```

O parâmetro ‘random_state’ serve para que os passos vistos aqui possam ser reproduzidos na íntegra, isso faz com que os dados fiquem sempre na mesma ordem.

Agora faremos concatenação dos dataframes dos gêneros em um único dataframe:

```
df_upsampled = pd.concat([df_Drama, df_Comedy], axis=0 )
df_upsampled = pd.concat([df_upsampled, df_Thriller ]
                         )
df_upsampled = pd.concat([df_upsampled, df_Action ]
                         )
df_upsampled = pd.concat([df_upsampled, df_Romance ]
                         )
df_upsampled = pd.concat([df_upsampled, df_Crime ]
                         )
df_upsampled = pd.concat([df_upsampled, df_Science_Fiction ]
                         )
df_upsampled = pd.concat([df_upsampled, df_Horror ]
                         )
df_upsampled = pd.concat([df_upsampled, df_Family ]
                         )
df_upsampled = pd.concat([df_upsampled, df_Fantasy ]
                         )
```

Vejamos as quantidades por gênero novamente:

```
df_upsampled['genres'].value_counts()
```

```
War               2297
History           2297
Animation         2297
Family            2297
Thriller          2297
Comedy            2297
Crime             2297
Science Fiction   2297
Drama             2297
Romance           2297
Western           2297
Mystery           2297
Horror            2297
Fantasy           2297
Action            2297
Documentary       2297
Foreign           2297
TV Movie          2297
Music             2297
Name: genres, dtype: int64
```

O porquê de cada gênero ter que estar em um array é devido ao uso do módulo 'MultiLabelBinarizer' do scikit-learn, caso o gênero não esteja em um array, não é possível fazer a transformação reversa.

Agora que balanceamos as classes podemos fazer a transformação de string em arrays conforme abaixo:

```
filmes_balanceados2['genres'] = filmes_balanceados2['genres'].map(lambda x: x.split(','))
filmes_balanceados2['genres'].head()
```

```
0               [Action]
1            [Adventure]
2              [Fantasy]
3      [Science Fiction]
4            [Adventure]
              ...
12155          [Comedy]
12156           [Drama]
12157         [Romance]
12158        [TV Movie]
12159      [Documentary]
Name: genres, Length: 12160, dtype: object
```

O último tratamento de dados antes de começar a criação dos modelos de machine learning é a retirada de strings não úteis do que será processado.

Criamos um método de processamento que pode ser visto na figura abaixo:

```
punctuation = """!()-[]{};:'"\, <>./?@#$%^&*_~"""

words = stopwords.words("english")
lemma = nltk.stem.WordNetLemmatizer()
def pre_process(text):
    text = str(text)
    remove_hyperlink = re.sub('http://\S+|https://\S+', '', text)
    for elements in remove_hyperlink:
        if elements in punctuation:
            remove_hyperlink = remove_hyperlink.replace(elements, " ")
    tokens = word_tokenize(remove_hyperlink)
    remove_words = [word for word in tokens if not word in words]
    text = [lemma.lemmatize(word) for word in remove_words]
    joined_words = " ".join(text)
    return joined_words
```

Com esse método processaremos todas as colunas texto que poderemos trabalhar:

```

title_df['overview'] = title_df['overview'].apply(pre_process)
filmes_balanceados['overview'] = filmes_balanceados2['overview'].apply(pre_process)
df_upsampled['overview'] = df_upsampled['overview'].apply(pre_process)

```

Por fim colocamos os gêneros que estavam como strings, de volta em arrays:

```
df_upsampled['genres'] = df_upsampled['genres'].map(lambda x: x.split(','))
```

```
title_df.head()
```

	overview	genres	title
0	In 22nd century paraplegic Marine dispatched m...	[Action, Adventure, Fantasy, Science Fiction]	Avatar
1	Captain Barbossa long believed dead come back ...	[Adventure, Fantasy, Action]	Pirates of the Caribbean: At World's End
2	A cryptic message Bond ' past sends trail unco...	[Action, Adventure, Crime]	Spectre
3	Following death District Attorney Harvey Dent ...	[Action, Crime, Drama, Thriller]	The Dark Knight Rises
4	John Carter war weary former military captain ...	[Action, Adventure, Science Fiction]	John Carter

```
filmes_balanceados.head(3)
```

	overview	genres	title
0	In 22nd century paraplegic Marine dispatched m...	[Action, Adventure, Fantasy, Science Fiction]	Avatar
1	In 22nd century paraplegic Marine dispatched m...	[Adventure, Fantasy, Action]	Pirates of the Caribbean: At World's End
2	In 22nd century paraplegic Marine dispatched m...	[Action, Adventure, Crime]	Spectre

```

df_upsampled.head(3)
```

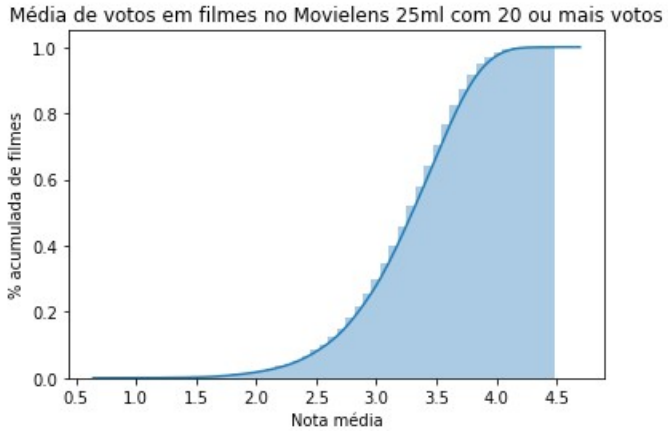
	overview	genres	title
12	Following death District Attorney Harvey Dent ...	[Drama]	The Dark Knight Rises
74	In 1933 New York overly ambitious movie produc...	[Drama]	King Kong
76	84 year later 101 year old woman named Rose De...	[Drama]	Titanic

Para o dataset do movielens foi feito um processo idêntico.

4. Análise e Exploração dos Dados

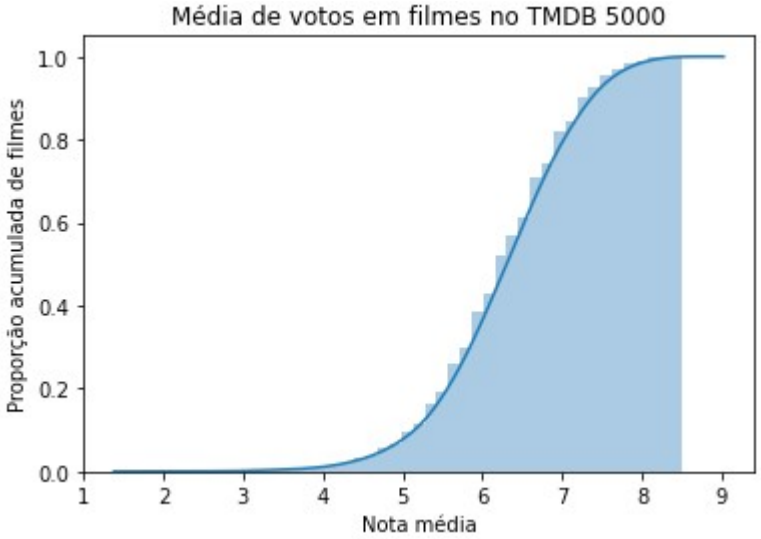
4.1. Visualizando a CDF (Cumulative Distribution Functions)

Continuando com a análise das notas dos datasets do movielens e do TMDB veremos agora a função de distribuição cumulativa das médias dos filmes nos datasets.



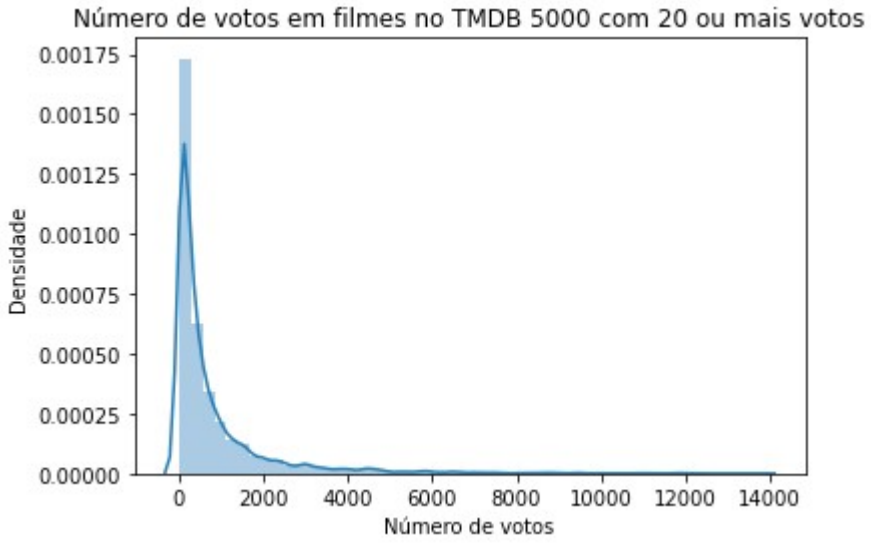
Nesse gráfico podemos perceber que aproximadamente 50% dos filmes tem uma nota média menor que 3,5. Se o filme Toy Story tem uma média 3,9, ele está acima de 80% dos filmes do conjunto. Já um filme com nota 2,5 estaria abaixo de 90% dos filmes!

Vamos repetir esse processo para os dados do TMDB 5000:



Lembrando que no movielens a média vai de 0 a 5 e no TMDB vai de 0 a 10. Se tentarmos fazer uma normalização grosseira nas notas do TMDB dividindo as notas por 2, vemos que as notas nos dois datasets têm comportamentos bem semelhantes

4.2. Visualizando outras informações



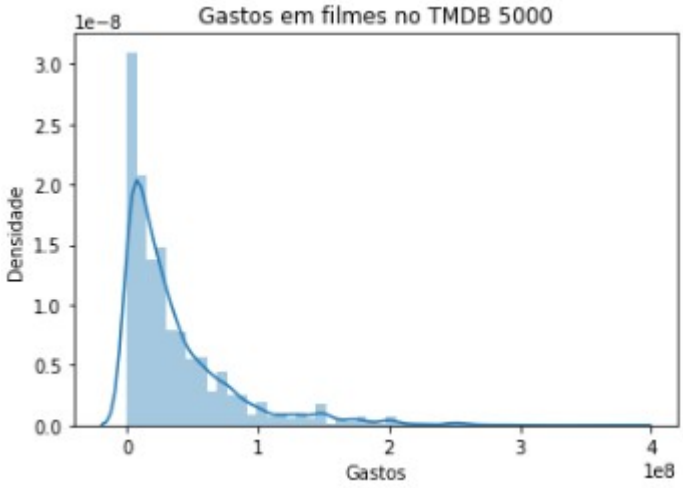
De acordo com o que fizemos na breve análise estatística, onde retiramos os filmes com menos de 20 votos e plotando esses votos, temos um histograma que, visualmente, lembra uma distribuição gama, tendo um comportamento bem diferente dos histogramas que geramos anteriormente.

Faz algum sentido, pois quanto mais popular um filme, maior a quantidade de votos que ele terá, e o número de filmes populares certamente é pequeno. No gráfico, isso se traduz no fato de que temos bem mais filmes com menos de 1000 votos do que, por exemplo, filmes com mais de 6000.

Será que podemos repetir esse processo para as outras colunas - por exemplo, o orçamento (budget)? Se listarmos essa coluna, veremos que vários registros apresentam o valor 0. Ou seja, provavelmente existem filmes nesse conjunto para os quais essa informação não está disponível, tendo sido preenchidos com esse valor.

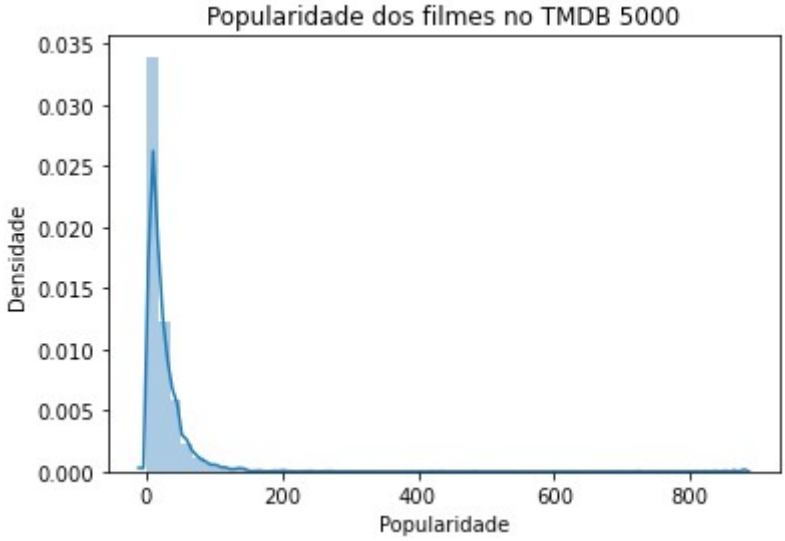
No nosso histograma, desejamos manter somente os filmes cujo orçamento seja maior do que 0.

Segue a distribuição do orçamento em um histograma



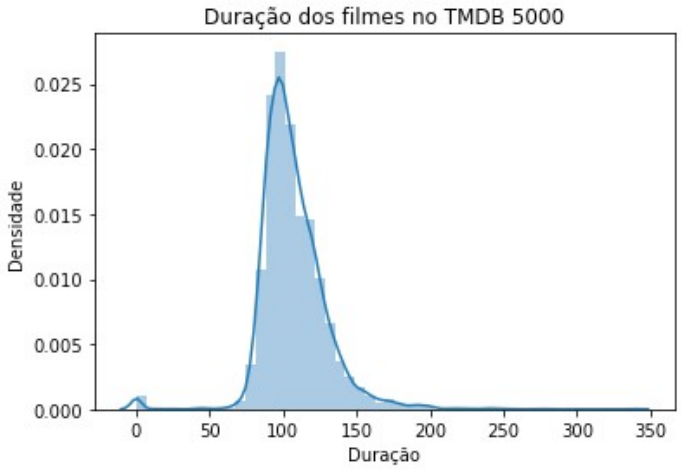
O gráfico de orçamento tem o mesmo comportamento que o gráfico de número de votos. Perceba que os números do eixo x estão multiplicados por 10^8 ou 100.000.000.

Vamos ver a coluna de popularidade(popularity). Executando a query ‘tmdb.query("popularity == 0").shape’ no pandas, temos que uma linha tem 0 de popularidade, como não vai afetar o gráfico, vamos considerar que essa linha tem realmente popularidade 0. Vejamos o gráfico de densidade de popularidade:

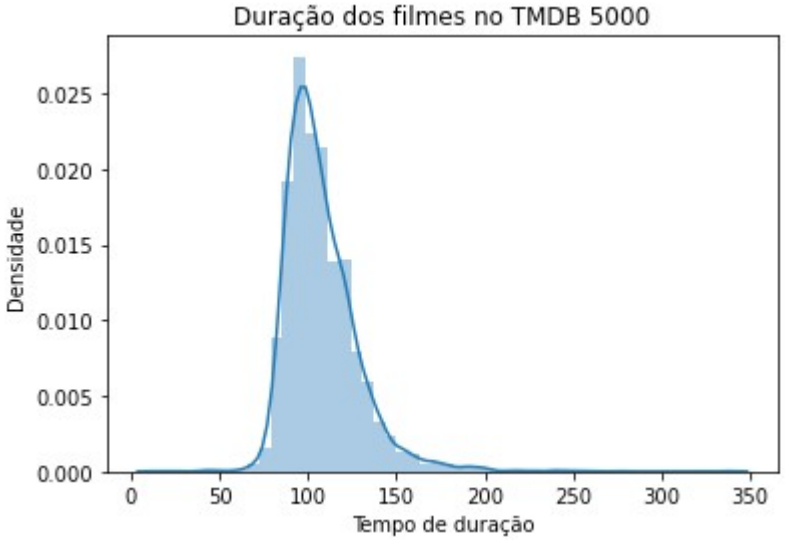


O comportamento da popularidade é o mesmo dos comportamentos encontrados nos gráficos anteriores. Assim podemos ver que poucos filmes têm grande popularidade.

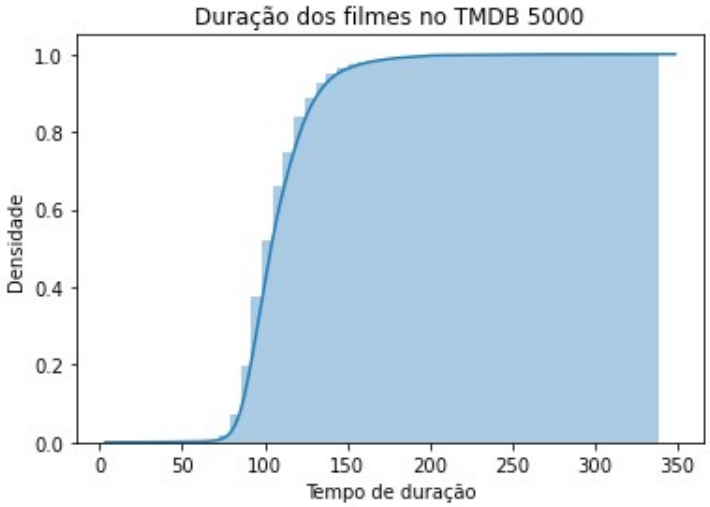
Em seguida, veremos a coluna de duração do filme (runtime). Quando tentamos gerar o gráfico, nos é retornado um erro informando que não é possível converter float NaN para integer. Isso nos indica que existem valores que não são números e não podem ser convertidos para integer, ou simplesmente indica que existem valores nulos na coluna. Rodando o comando tmdb.runtime.isnull().sum(), vemos que temos 2 linhas com valores nulos. Passando a coluna runtime sem os valores nulos nos possibilita gerar o seguinte gráfico:



Podemos perceber no gráfico anterior que temos um pico de valores 0, indicando que temos vários filmes onde esse campo foi colocado com o valor 0. Plotaremos um novo gráfico removendo esses valores utilizando a query query("runtime>0")



Aproveitando, vamos ver o gráfico cumulativo de durações de filme:



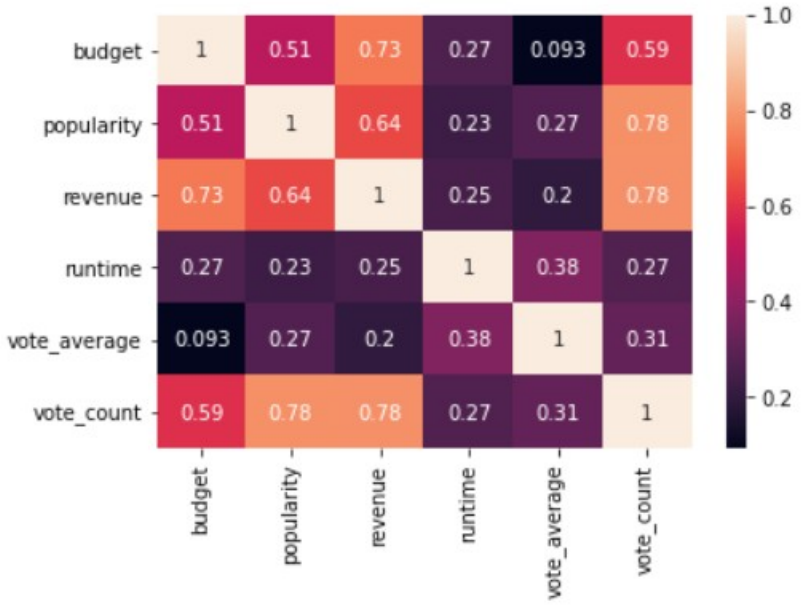
Podemos inferir visualmente, que cerca de 40% dos filmes no conjunto de dados do TMBD têm menos de 100 minutos de duração. Mas se, por exemplo, quiséssemos saber que valor separa 80% desse conjunto? A função quantile nos traz a resposta:

```
tmdb.query("runtime>0").runtime.dropna().quantile(q=0.8)
```

```
121.0
```

Assim, sabemos que 80% dos nossos filmes têm 121 minutos ou menos, e que 20% têm uma duração maior.

Agora vamos ver um gráfico nos responde a seguinte pergunta: Existe correlação entre as variáveis numéricas do TMDB 500 ?



Pelo gráfico acima podemos ver que, por exemplo, budget, revenue, popularity e vote_count são bem correlacionados

4.3. O efeito do tamanho de uma amostra

Para continuar com nossas análises vamos ver a média das médias dos filmes do movielens:

```
average_vote_min20.mean()
```

```
3.237997737700967
```

Sabendo do fato de que a nota mínima no movielens é 0,5 e a máxima é 5, poderíamos esperar que a média das médias fosse 2.75. Portanto, em geral, parece que os usuários atribuem notas maiores que 2.75 aos filmes.

Será que se de repente, utilizarmos somente 10% da amostra, poderíamos generalizar a média para a amostra inteira?

Vamos ver para os 5 primeiros registros:

```
average_vote_min20[0:5].mean()
```

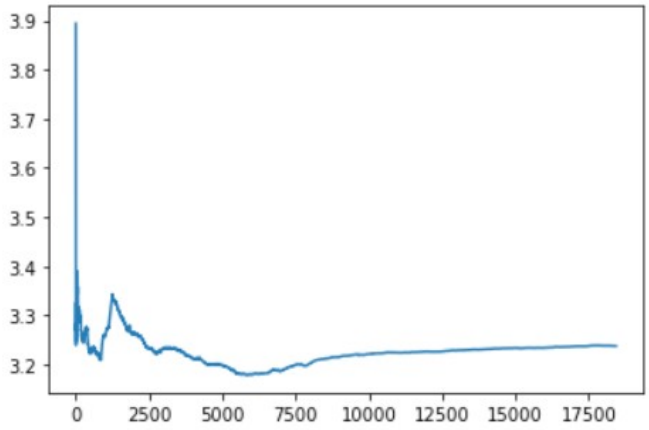
```
3.2398489591223765
```

Quantos registros temos no dataset em trabalho?

```
len(average_vote_min20)
```

```
18430
```

Vamos plotar essas médias em um gráfico para que possamos analisar o seu comportamento. Então começaremos com 1 amostra e vamos acrescentando elementos da amostra em nosso conjunto e plotaremos como a média se comporta à medida que o conjunto cresce

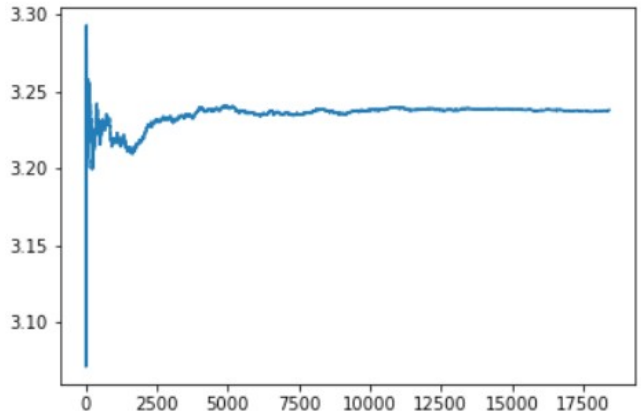


Perceba que, no começo, a média tem quedas e elevações abruptas. Porém, em determinado momento do conjunto (por volta dos 10000 registros), a distribuição parece se estabilizar, mantendo-se quase constante.

Isso acontece pois, quando temos poucos elementos, cada novo registro no conjunto faz uma diferença bastante significante. Porém, conforme a amostra vai se tornando maior, os novos valores passam a interferir menos no resultado final.

Isso significa que, se tivéssemos uma amostra muito pequena, ou mal escolhida, poderíamos ter chegado a resultados diferentes e que levassem a outras interpretações a respeito desses dados. Com 2000 votos, por exemplo, temos uma média de 3.35, um valor mais alto que a média final de 3.23.

Repare, também, que esses votos estão em uma ordem específica, fornecida pelo próprio dataset, e não sabemos se essa ordem tem alguma característica oculta que possa estar afetando a distribuição. Vamos testar isso aleatorizando a ordem do nosso conjunto.



Dessa vez, nosso gráfico começa com valores altos, diminui bastante e então progride de maneira mais estável, tornando-se muito menos sensível à entrada de novas médias.

4.4. Intervalo de confiança com t-Student e com ztest

Escolhendo os dados de maneira simples, pudemos ver que dependendo do tamanho da amostra e como os dados são escolhidos, essa escolha irá interferir na análise de dados.

Vamos tentar entender, então, qual o intervalo de confiança que temos para as médias dos nossos filmes. Isso não quer dizer que todos os filmes do universo de dados terão uma média 3.23, que é a média dos filmes do TMDB 5000 que apresentam pelo menos 20 votos, mas queremos criar uma generalização, a partir dessa amostra, sobre os filmes que existem no universo.

O objetivo, então, é utilizarmos o Teste Z para encontrarmos um intervalo de confiança que abrangerá não só os filmes da nossa amostra, mas também aqueles fora dela. Para isso, usaremos a função zconfint() (que se refere ao intervalo de confiança no Teste Z), passando como parâmetro os nossos dados (average_vote_min20). O parâmetro alpha, que é o valor de p, já é previamente configurado como 0.05 (5%).

```
from statsmodels.stats.weightstats import zconfint
#movie lens
zconfint(average_vote_min20)
```

```
(3.2310351155994366, 3.244960359802507)
```

```
from statsmodels.stats.weightstats import zconfint
#tmdb
zconfint(tmdb_min_20.vote_average          )
```

```
(6.238014699215095, 6.290598157237297)
```

Ou seja, acreditando que nosso conjunto de dados seja uma amostra grande suficiente para a aplicação desse tipo de teste (que exige normalidade e uma grande quantidade de dados), teremos um intervalo de confiança entre 3.2310 e 3.2449 para os filmes em geral, e não só para os da amostra.

Nos cursos de estatística, aprendemos que existem outros tipos de testes, um exemplo seria o Teste T, e existem diferenças primordiais entre eles. Ambos exigem algumas características na nossa distribuição ou nos nossos dados, mas, em linhas gerais, utilizamos o Teste Z quando temos um conjunto maior, e o Teste T quando esse conjunto é menor.

Como nossa amostra pode ser considerada grande, executaremos o Teste T apenas para verificarmos o resultado.

```
descr_todos_com_10_votos.tconfint_mean()
```

```
(3.2310346582845444, 3.2449608171173994)
```

O intervalo de confiança utilizando o Teste T é muito parecido com aquele que encontramos no Teste Z. Lembrando que cada teste deve ser utilizado em situações distintas, de acordo com a necessidade.

4.5. Teste Z para uma amostra

Já sabemos que a média dos nossos dados é aproximadamente 3.2398, e aplicamos o Teste Z para obter um intervalo de confiança (com alpha = 0,5%) entre 3.2310 e 3.2449. Esse intervalo compreenderia a média de todos os filmes no universo, supondo que eles tenham características similares ao conjunto do TMBD 5000.

Agora, vamos analisar um filme específico no nosso conjunto, neste caso, o primeiro registro do dataset, comparando sua média com os outros valores que obtemos.

```
filmes = pd.read_csv("D:\\TCCDataScience\\dados\\movielens\\ml-25m\\movies.csv")
filmes.query("movieId==1")
```

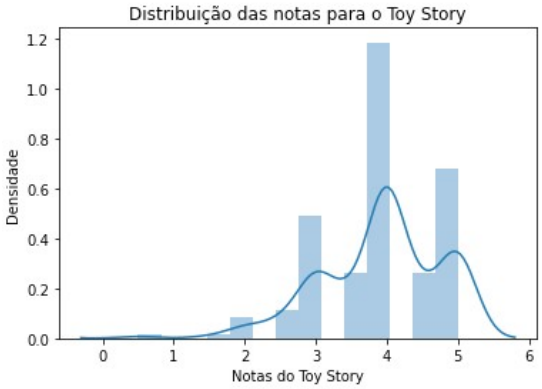
	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

Uma amostra das notas do filme

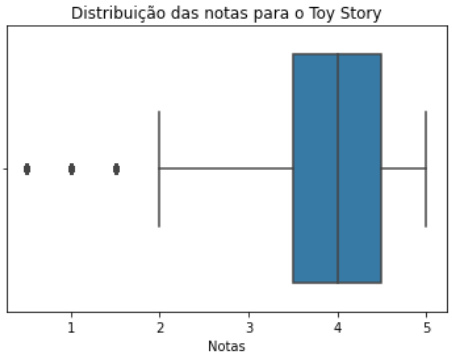
```
notas1 = notas.query("movieId ==1")
notas1.head()
```

	userId	movieId	rating	timestamp
70	2	1	3.5	1141415820
254	3	1	4.0	1439472215
910	4	1	3.0	1573944252
1152	5	1	4.0	858625949
1304	8	1	4.0	890492517

Vamos verificar a distribuição de notas para o filme



No movielens as notas variam de 0 a 5, com gradações de 0.5 (0, 0.5, 1, 1.5, etc). Como os valores não são distribuídos de forma contínua, vamos dar uma olhada também no boxplot o conjunto



Essa visualização nos permite interpretar melhor a distribuição dos nossos dados. Por exemplo, a mediana está posicionada na nota 4 e parece existir uma densidade semelhante nos dois quartis imediatamente adjacentes. Como 5 é a nota máxima, as notas não se distribuem muito para a direita, ao contrário do que acontece para a esquerda.

Visualmente, a mediana 4 do Toy Story já é bastante superior à mediana 3,2992 que tínhamos na distribuição das notas médias dos filmes do movielens com 20 ou mais votos. Porém, será que essa percepção se sustenta ? Afinal, pode ser que os nossos dados estejam enviesados.

Mesmo que fossem três pessoas completamente aleatórias, ainda existiria uma chance de que todas elas dessem a mesma nota, e as nossas informações seriam, então, fruto do acaso.

Vamos calcular a média de notas do Toy Story:

```
notas1.rating.mean()
```

```
3.893707794587238
```

Realmente, a média dos dados que coletamos é maior do que a média dos filmes com pelo menos 20 votos, que era 3.2379. Mas será que essa diferença é real? Ou ela é fruto do acaso?

Para termos uma análise mais objetiva dessa média, já que temos mais de 30 registros no nosso conjunto, podemos aplicar o Teste Z:

```
zconfint(notas1.rating)
```

```
(3.886162842774441, 3.9012527464000346)
```

Com alpha = 0.5, temos um intervalo de confiança variando entre aproximadamente 3.8861 e 3.9012 da média desse filme no mundo real, e não só nos dados que coletamos. Esse intervalo é mais alto que a média dos filmes com pelo menos vinte votos, mas talvez comparar um intervalo com um número fixo não seja realmente viável.

Também podemos realizar outro teste com as nossas médias, tentando afirmar que a nota média do Toy Story na verdade é a nota média que encontramos para os filmes com pelo menos vinte votos. Vamos fazer um novo teste Z que nos retorna um pvalue confirmando se, no mundo real, a média desse filme seria ou não igual a 3.237997737700967 (a nota média do movielens).

```
from statsmodels.stats.weightstats import ztest
ztest(notas1.rating, value = 3.237997737700967)
```

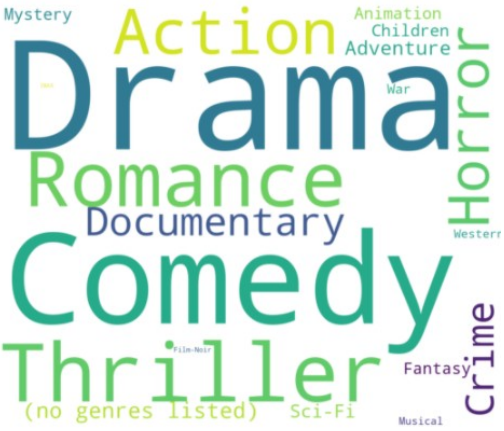
```
(170.33483151185402, 0.0)
```

O ztest() nos devolve duas variáveis: o valor estatístico e o pvalue. Repare que este último é 0, e menor que o nosso 0.05 (o alpha). Portanto, podemos descartar a hipótese de que, no mundo real, a média do Toy Story é igual a 3.237997737700967 (também chamada de hipótese nula).

Quando trabalhamos com bigdata, acabamos caindo nesse tipo de problema, e torna-se mais interessante analisarmos não só a possibilidade dessa igualdade, como também o intervalo de confiança e as visualizações gráficas das nossas distribuições.

4.5. Algumas visualizações

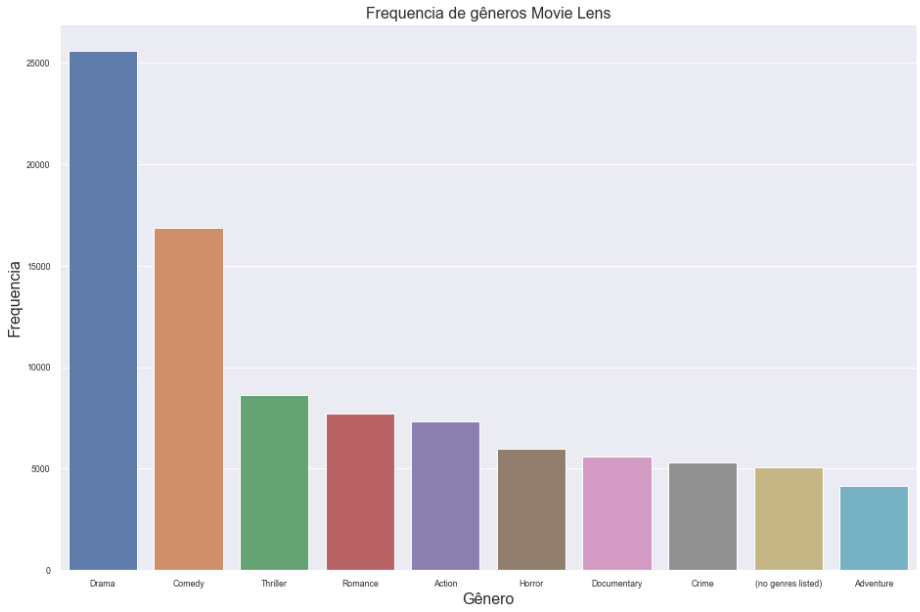
4.5.1. Nuvens de Gêneros



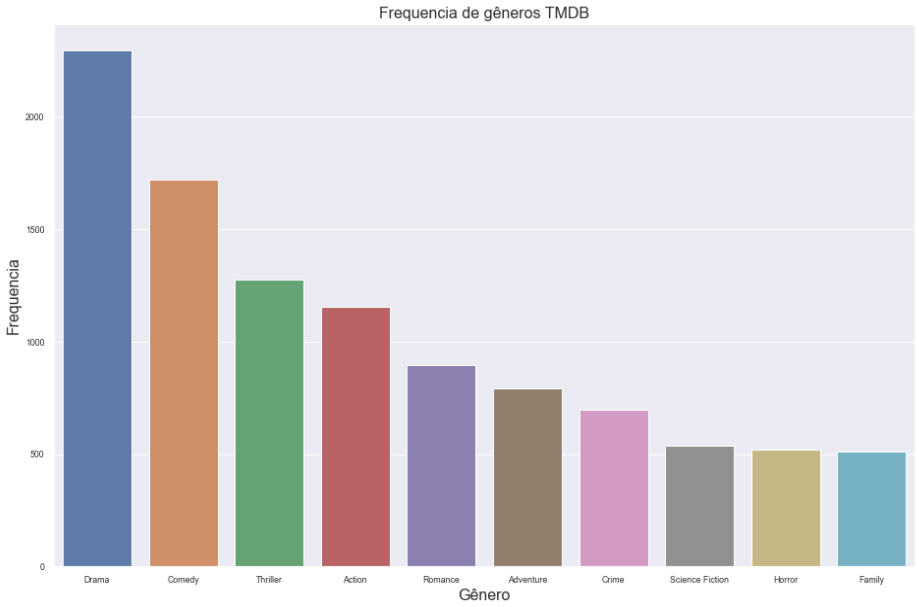
movielens

tmdb

4.5.2. Frequência de Gêneros de filmes



Top 10 gêneros no Movielens



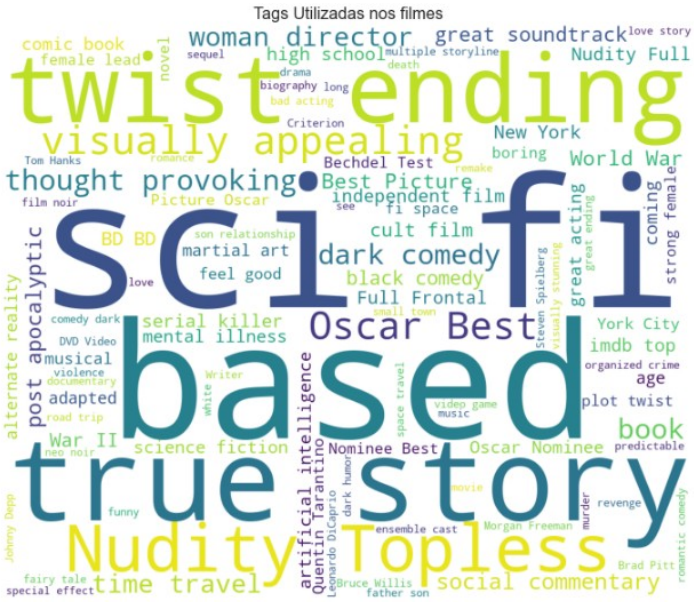
Top 10 gêneros no TMDB

4.5.3. Filmes com as notas mais altas

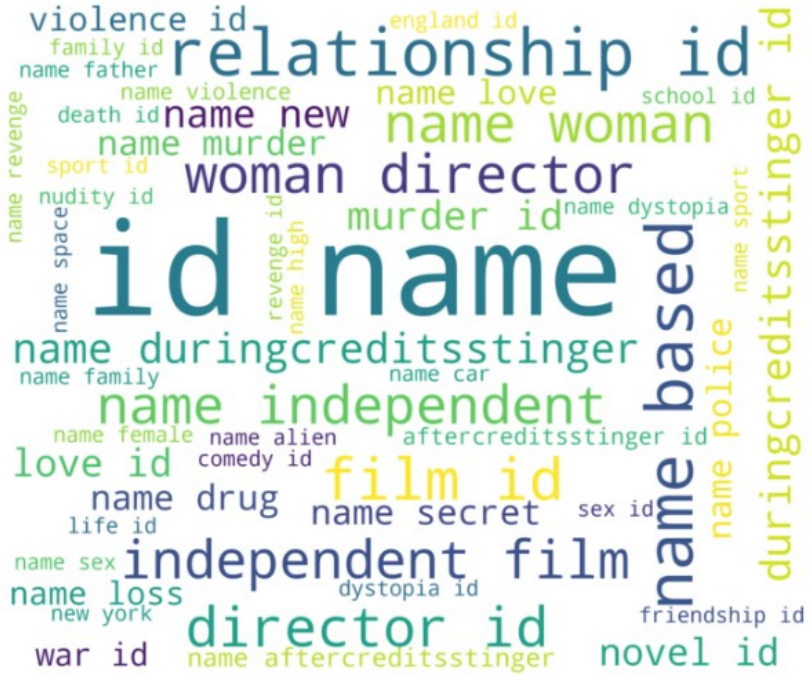
```
#Filmes com  as maiores notas
top_10_rated = movies.query("qtd_votos >= 50").sort_values(by ='rating', ascending=False )[:10]
top_10_rated
```

	movieId	title	genres	rating	qtd_votos
45741	171011	Planet Earth II (2016)	Documentary	4.483096	1124.0
40597	159817	Planet Earth (2006)	Documentary	4.464797	1747.0
314	318	Shawshank Redemption, The (1994)	Crime|Drama	4.413576	81482.0
45593	170705	Band of Brothers (2001)	Action|Drama|War	4.398599	1356.0
45973	171495	Cosmos	(no genres listed)	4.326715	277.0
840	858	Godfather, The (1972)	Crime|Drama	4.324336	52498.0
49559	179135	Blue Planet II (2017)	Documentary	4.289833	659.0
49	50	Usual Suspects, The (1995)	Crime|Mystery|Thriller	4.284353	55366.0
58237	198185	Twin Peaks (1989)	Drama|Mystery	4.267361	288.0
1190	1221	Godfather: Part II, The (1974)	Crime|Drama	4.261759	34188.0

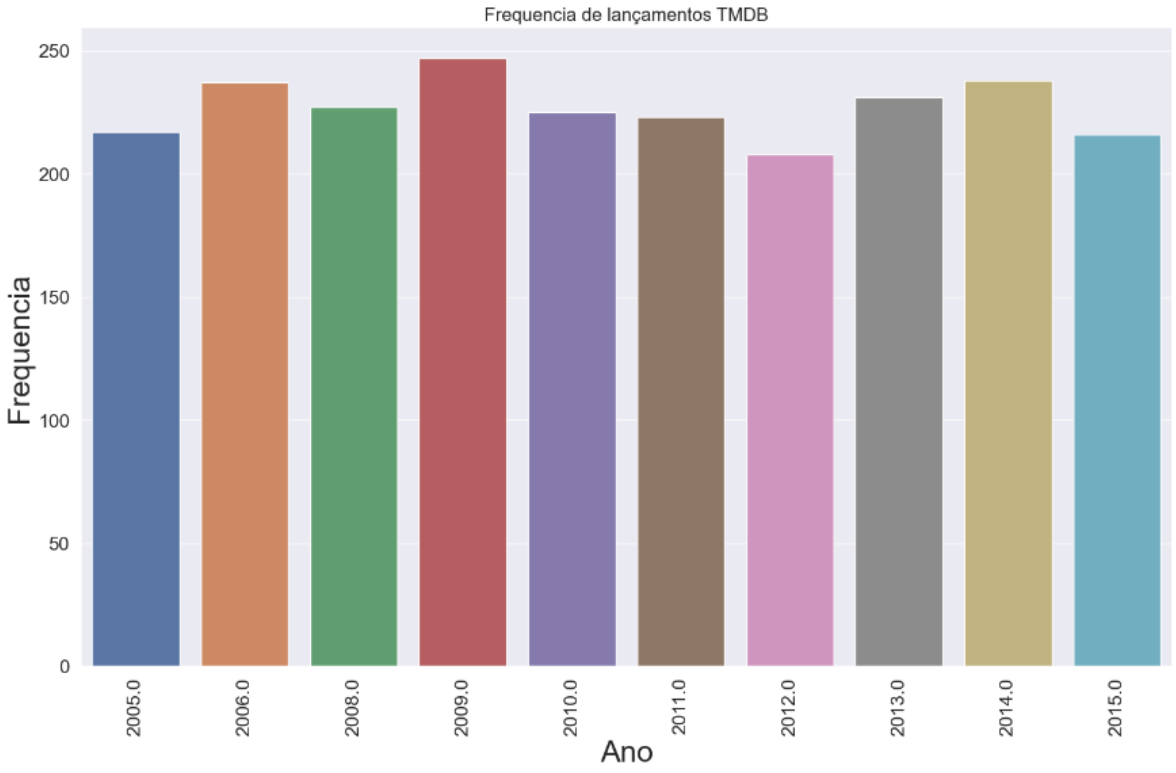
4.5.4. Nuvem de tags do movielens



4.5.5. Nuvem de keywords do TMDB

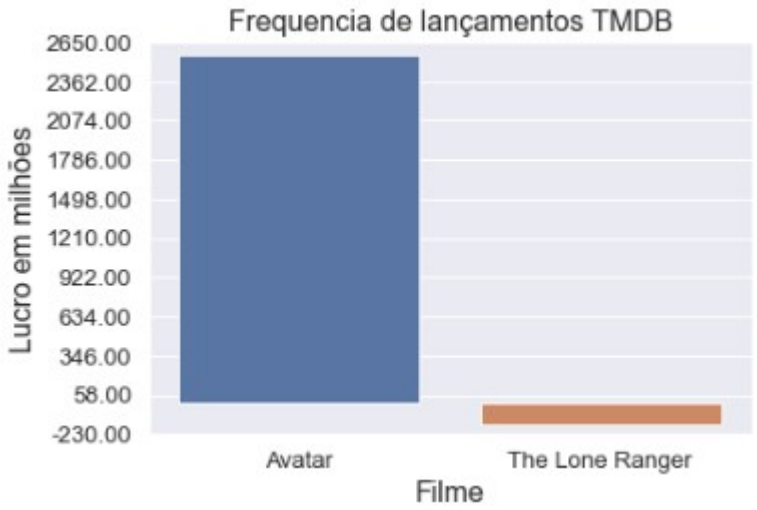


4.5.6. Qual o ano em que houve mais lançamentos de filmes




```
1 2009 foi o ano com mais lançamentos de filmes.
```

4.5.7. Qual filme teve maior faturamento? Qual teve o maior prejuízo?



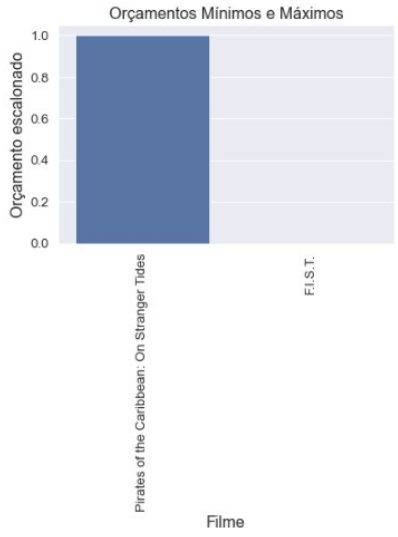
	original_title	profit
0	Avatar	2550965087
13	The Lone Ranger	-165710090

Conferir as maiores bilheterias em: https://www.omelete.com.br/marvel-cinema/vingadores-ultimato-endgame/vingadores-ultimato-10-maiores-bilheterias-da-historia#20

4.5.8. Qual filme teve maior orçamento? E o menor?

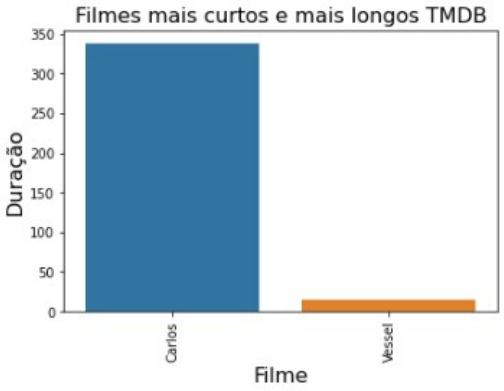
	original_title	budget	scaled_budget
17	Pirates of the Caribbean: On Stranger Tides	380000000	1.0
2933	F.I.S.T.	11	0.0

Orçamento em milhões de dólares

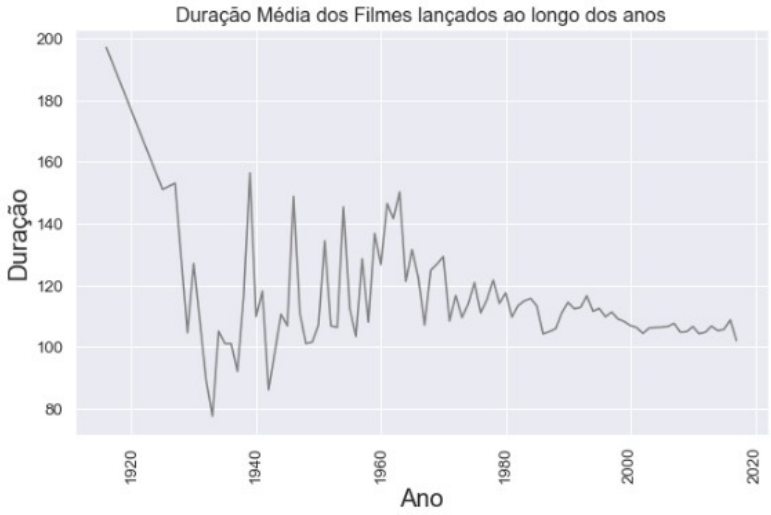


Orçamento escalonado

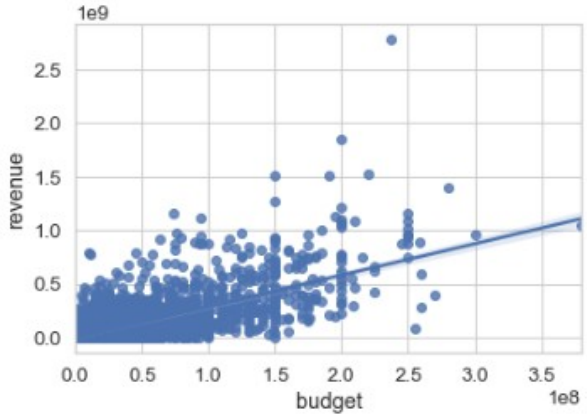
4.5.9. Qual filme com a maior duração? E o filme com a menor?



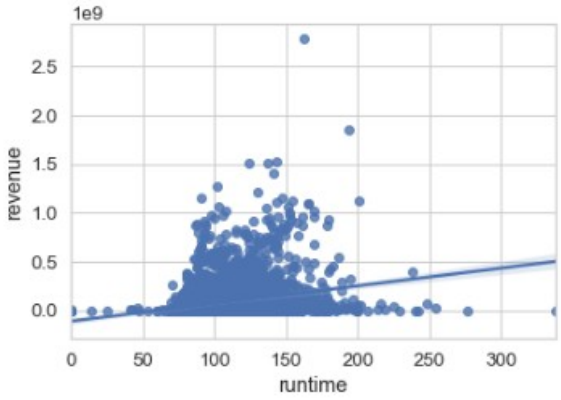
4.5.10. Qual a duração média dos filmes ao longo dos anos?



4.5.11. Qual o relacionamento entre revenue(faturamento), budget(orçamento) e tempo do filme(runtime)?

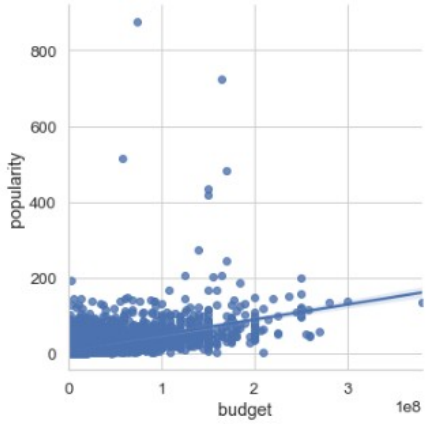


O faturamento tem uma relação positiva com o orçamento (correlação forte: coeficiente de correlação de 0.73)

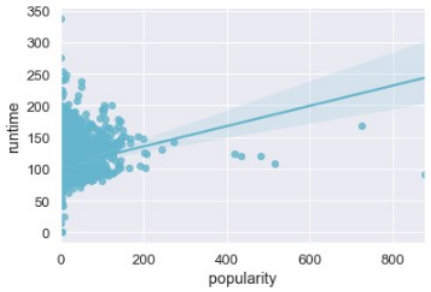


O faturamento não tem uma relação clara com a duração do filme (correlação fraca: coeficiente de correlação de 0.25)

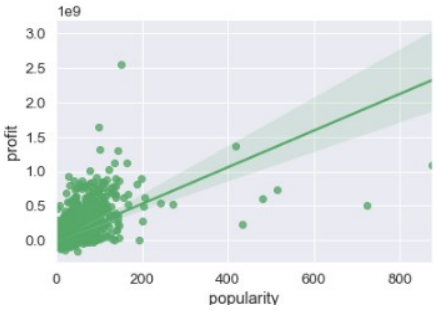
4.5.12. Qual o efeito de revenue(faturamento), budget(orçamento) e tempo do filme(runtime) com a popularidade de um filme?



Aparentemente, quanto maior o custo de um filme maior sua popularidade (correlação moderada: coeficiente de correlação de 0.51)



O tempo do filme parece não afetar a popularidade de um filme (correlação fraca: coeficiente de correlação de 0.23)



Quanto mais popular um filme, maior o lucro obtido (correlação moderada/forte: coeficiente de correlação de 0.64)

5. Criação de Modelos de Machine Learning

A construção dos modelos de machine learning foi feita em python no jupyter, utilizando o scikit-learn (https://scikit-learn.org/) e o Natural Language Toolkit (https://www.nltk.org/).

Neste trabalho tentamos fazer um classificador de gêneros de filmes de acordo com o 'overview' fornecido pelo dataset do TMDB. No caso do movielens, utilizando o dataset 'links.csv' que tem os identificadores dos filmes do movielens no TMDB e utilizando a API do TMDB conseguimos um dataset de mais de 57.000 registros. Vamos ver se os resultados dos mesmos algoritmos e parâmetros darão o mesmo resultado que no dataset do TMDB que tem perto de 5.000 registros.

Além dos tamanhos dos datasets diferentes, ainda vamos tentar tratar o problema de classes desbalanceadas. Nos trabalhos que encontramos facilmente na internet temos modelos de classificação com 2 a 5 classes, em nosso trabalho teremos 19 classes, onde cada classe é um gênero do filme.

Dentre as diversas técnicas de tratamento de classes desbalanceadas trabalharemos com:

- upsample – Esse método consiste em duplicar os registros das classes minoritárias randomicamente para que todas as classes tenham a mesma quantidade de registros que a classe com maior frequência de ocorrências. Aqui usaremos o módulo de resampling do Scikit-Learn.
- Mudança de métrica de performance – Aqui utilizaremos a acurácia e o score F1 para cada algoritmo.
- Penalizar Algoritmos – Aqui com um parâmetro, o algoritmo penaliza erros nas classes de menor ocorrência na proporção em quão pouco representativa aquela classe é.
- Utilizar algoritmos baseados em árvores – No trabalho utilizaremos alguns algoritmos baseados em árvores

Vamos tentar trabalhar com o parâmetro balanced setado, pois assim vamos ter uma base para os três conjuntos de dados que trataremos para cada dataset. A seguir descrevemos que 3 conjuntos de dados são esses

5.1 Conhecendo os dados utilizados

Tanto para o TMDB como para o movieLens, para os modelos de machine learning serão utilizadas as colunas: overview e genres. O nosso objetivo é que a partir de um overview de um filme o algoritmo de machine learning consiga prever a que gênero aquele filme pertence.

Amostra de dados do TMDB:

	overview	genres	title
0	In the 22nd century, a paraplegic Marine is di...	[Action, Adventure, Fantasy, Science Fiction]	Avatar
1	Captain Barbossa, long believed to be dead, ha...	[Adventure, Fantasy, Action]	Pirates of the Caribbean: At World's End
2	A cryptic message from Bond's past sends him o...	[Action, Adventure, Crime]	Spectre

Como podemos ver na figura acima, os gêneros são organizados em um array. Assim, para esse caso inicial, o número de classes é a combinação dos gêneros atribuídos a um filme, isso nos dá 1639 classes. Esse conjunto inicial de dados iremos chamar de '*sem tratamento*'

A partir desse conjunto inicial duplicamos a linha do filme para cada gênero atribuído. Os dados ficaram como abaixo:

	overview	genres	title
12157	"Signed, Sealed, Delivered" introduces a dedic...	Romance	Signed, Sealed, Delivered
12158	"Signed, Sealed, Delivered" introduces a dedic...	TV Movie	Signed, Sealed, Delivered
12159	Ever since the second grade when he first saw ...	Documentary	My Date with Drew

Após esse tratamento inicial iremos chamar esse conjunto de '*tratamento parcial*'. Esse tratamento nos fez perceber o problema das classes desbalanceadas. Quando contamos as ocorrências para cada gênero temos a seguinte contagem:

```
1 filmes_balanceados2['genres'].value_counts()
```

```
Drama              2297
Comedy             1722
Thriller           1274
Action             1154
Romance             894
Adventure           790
Crime               696
Science Fiction     535
Horror              519
Family              513
Fantasy             424
Mystery             348
Animation           234
History             197
Music               185
War                 144
Documentary         110
Western              82
Foreign              34
TV Movie              8
Name: genres, dtype: int64
```

Na figura acima podemos notar que os 19 gêneros presentes não estão balanceados, ou seja, temos uma quantidade muito diferente de ocorrências para cada gênero. Aqui utilizaremos a técnica de upsample para que todos os gêneros tenham a mesma frequência que o gênero ‘drama’. A técnica consiste em criar um dataframe para cada gênero e utilizar o resample da biblioteca Scitik-Learn. Após essa transformação as contagens ficaram assim:

```
Animation          2297
Comedy             2297
Fantasy            2297
Action             2297
Science Fiction    2297
Romance            2297
Foreign            2297
TV Movie           2297
Drama              2297
War                2297
Mystery            2297
Documentary        2297
Horror             2297
Western            2297
Thriller           2297
Crime              2297
Family             2297
History            2297
Music              2297
Name: genres, dtype: int64
```

O conjunto acima chamaremos de ‘*upsampled*’. Então, além de compararmos como diversos modelos de machine learning se comportam para essa classificação, iremos passar os algoritmos nos três conjuntos de dados, a saber: ‘Sem tratamento’, ‘Tratamento Parcial’ e ‘upsampled’

Como faremos processamento de linguagem natural, precisamos fazer mais alguns preprocessamentos nos conjuntos de dados antes de submetê-los aos modelos de machine learning. Os preprocessamentos são:

- Remoção de pontuação e das ‘*stop words*’ do overview dos filmes
- Criação de uma matriz binária(amostras x classes) que indica a presença de um label da classe
- Criação da TF-IDF para a coluna ‘overview’ para cada conjunto de dados. TF-IDF (abreviação do inglês *term frequency–inverse document frequency*, que significa frequência do termo–inverso da frequência nos documentos)

5.2 Algoritmos utilizados

Antes de vermos os algoritmos, vamos falar do módulo de cross validation utilizado para tentativa de otimização dos algoritmos utilizados. O módulo de cross-validation utilizado foi o GridSearchCV. Além desse módulo, foi utilizado também o módulo de pipeline. Como estamos utilizando o módulo de processamento de linguagem natural, o módulo de processamento de linguagem natural também entrou no cross validation. No script final vamos ter o código do cross-validation comentado, pois o GridSearchCV faz testes exaustivos com todos os parâmetros passados com a base de dados fornecida, alguns algoritmos levaram mais de 12h para finalizar o treinamento. Assim, no final foram definidos os dados otimizados fornecidos pelo GridSearchCV para a coleta final das métricas. Segue um exemplo de como funciona o GridSerchCV. Vamos ver em duas partes, a primeira parte é a parte onde setamos os parâmetros a serem testados e a segunda é o código onde o GridSearchCV faz o treinamento em si com os diversos parâmetros passados.

A primeira parte:

```
1
2  nltk.download('stopwords')
3  words = stopwords.words("english")
4  pipeline = Pipeline([
5  ('tfidf', TfidfVectorizer(stop_words=words)),
6  ('clf', OneVsRestClassifier(LogisticRegression( ))),
7
8  ])
9  # parameters = {
10 #     'tfidf__ngram_range': [(1, 1), (1, 2), (1, 3)],
11 #     'tfidf__max_features': [10000,15000,20000,25000],
12 #     "clf__estimator__C": [0.01, 0.1, 1],
13 #     "clf__estimator__class_weight": ['balanced'],
14 #     'clf__estimator__solver':['newton-cg',   'liblinear', 'sag', 'saga'],
15 #     'clf__estimator__multi_class':[ 'ovr', 'multinomial'],
16 # #     'clf__estimator__penalty': ['l1', 'l2', 'elasticnet' ],
17 # }
18 parameters = {
19    'tfidf__ngram_range': [ (1, 2)],
20    'tfidf__max_features': [15000],
21    "clf__estimator__C": [ 1],
22    "clf__estimator__class_weight": ['balanced'],
23    'clf__estimator__solver':['newton-cg'],
24    'clf__estimator__multi_class':[ 'multinomial'],
25 #    'clf__estimator__penalty': ['l1', 'l2', 'elasticnet' ],
26 }
```

As duas primeiras linhas do trecho são para baixar as ‘stopwords’ em inglês que serão utilizadas pelo TfidfVectorizer. Nas linhas 4-8 definimos o que vai fazer parte do pipeline de treinamento. No nosso caso, no pipeline é incluído o TfidfVectorizer para transformação da coluna overview em uma matriz de features TF-IDF e um estimador OneVsRestClassifier com o algoritmo de Regressão Logistica (LogisticRegression). Nas linhas 18-26 foram setados os parâmetros a

serem utilizados no treinamento do algoritmo. Nesse trecho podemos ver que cada parâmetro está recebendo somente um valor. Como explicamos anteriormente, esses valores foram os valores finais utilizados para coleta de métricas.

Para entender a complexidade que termina exigindo muito poder de computação e faz com que certos algoritmos por já serem pesados e com a quantidade de parâmetros utilizados o treinamento termina por demorar bastante. No nosso exemplo que está comentado temos para cada linha:

- 3 parâmetros
- 4 parâmetros
- 3 parâmetros
- 1 parâmetro
- 4 parâmetros
- 2 parâmetros
- 3 parâmetros

O GridSearchCV faz um treinamento para cada combinação de parâmetros, no nosso caso fica: 3*4*3*1*4*2*3 = 864 treinamentos. Mas como veremos adiante, passo um parâmetro para que o GridSearchCV divida o conjunto de treinamento em 2, e o GridSearchCV faz um treinamento para cada conjunto de parâmetros com cada conjunto de treinamento isso nos totaliza 1728 treinamentos. Se levar em consideração que cada treinamento leve 1 min, teremos um treinamento total de 28,8 horas.

Segunda parte:

```
1
2 grid_search_tune = GridSearchCV(
3 pipeline, parameters, cv=2, n_jobs=2, verbose=3)
4 grid_search_tune.fit(train_x, ytrain )
5 print
6 print("Melhor conjunto de parâmetros:")
7 print (grid_search_tune.best_estimator_.steps)
```

```
Fitting 2 folds for each of 1 candidates, totalling 2 fits
```

Aqui estamos instanciando um objeto GridSearchCV passando para ele o pipeline e parâmetros definidos anteriormente. O parâmetro ‘cv’ indica em quantas partes o conjunto de dados deve ser divido. O parâmetros ‘n_jobs’ indica quantos processadores devem ser utilizados no treinamento e o parâmetro ‘verbose‘ indica quão detalhadas devem ser as mensagens de processamento. Na linha 4 instruímos o GridSearchCV a fazer o treinamento com os dados informados. E na linha 7 é impresso caraterísticas do melhor conjunto de parâmetros.

Exemplo de saída desse trecho de código:

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   2 out of   2 | elapsed:    2.4s remaining:    0.0s
[Parallel(n_jobs=2)]: Done   2 out of   2 | elapsed:    2.4s finished

Melhor conjunto de parâmetros:
[('tfidf', TfidfVectorizer(max_features=15000, ngram_range=(1, 2),
                stop_words=['i', 'me', 'my', 'myself', 'we', 'our', 'ours',
                            'ourselves', 'you', "you're", "you've", "you'll",
                            "you'd", 'your', 'yours', 'yourself', 'yourselves',
                            'he', 'him', 'his', 'himself', 'she', "she's",
                            'her', 'hers', 'herself', 'it', "it's", 'its',
                            'itself', ...])), ('clf', OneVsRestClassifier(estimator=LogisticRegression(C=1, class_
weight='balanced',
                                                    multi_class='multinomial',
                                                    solver='newton-cg')))]
```

Na saída é informado qual o melhor conjuntos de parâmetros a serem utilizados para cada elemento do pipeline.

Agora veremos os elementos utilizados em cada pipeline:

- TfidfVectorizer: Como dissemos anteriormente o TfidfVectorizer converte um documento em uma matriz de features TF-IDF. Os parâmetros foram praticamente os mesmos para todos os algoritmos. Os parâmetros utilizados:
  - stop_words: Conjunto de palavras que deve ser ignoradas durante o treinamento
  - max_features: Número que indica quantos dos elementos mais frequentes do vocabulário gerado devem ser utilizados
  - n_gram_range: Uma tupla que indica os valores máximos e mínimos de n_grams devem ser extraídos. No nosso exemplo foi utilizado a tupla (1,2), indicando que devem ser extraídos conjuntos de 1 ou duas palavras
- OneVsRestClassifier: Essa classe na verdade não é um estimador, mas uma estratégia para treinamento de algoritmos/estimadores onde cada classe é treinada contra todas as outras classes
- LogisticRegression: Classe que implementa a regressão logistica onde pode utilizar vários algoritmos diferentes de regressão. Parâmetros utilizados:
  - penalty: Qual normalização deve ser utilizada na penalização das classes
  - solver: Qual algoritmo deve ser utilizado
  - class_weight: Como foi utilizado o valor ‘balanced’, isso faz com que o peso de uma classe seja inversamente proporcional à frequência de ocorrências da classe
  - C: Regularizador dos pesos atribuídos às classes
  - multi_class: Indica se o treinamento deve treinar para cada classe ou para o conjunto inteiro. Tem o valor ‘auto’ que faz essa escolha automaticamente de acordo com o conjunto de dados passado.

- RidgeClassifier: Classificador que usa a regressão de Ridge.
  - Solver: Foi utilizado o valor auto
  - class_weight: Como foi utilizado o valor ‘balanced’, isso faz com que o peso de uma classe seja inversamente proporcional à frequência de ocorrências da classe
- RidgeClassifierCV: Classificador que usa a regressão de Ridge com um cross-validation embutido
  - class_weight: Como foi utilizado o valor ‘balanced’, isso faz com que o peso de uma classe seja inversamente proporcional à frequência de ocorrências da classe
  - cv: Igual ao GridSearchCV, divide os dados na quantidade do número informado. Aqui foi utilizado o valor 2
- ExtraTreesClassifier: Esta classe implementa um estimador meta que treina uma quantidade de árvoes de decisão randômicas (i.e. extra-trees) em vários sub-amostras do dataset e usa média para melhorar a acurácio preditiva e controle de over-fitting.
  - n_estimators: Indica o número de árvores na floresta
  - class_weight: Como foi utilizado o valor ‘balanced’, isso faz com que o peso de uma classe seja inversamente proporcional à frequência de ocorrências da classe
  - max_depth: A profundidade máxima da árvore
- NaiveBayes (MultinomialNB): Classificador para modelos multinomial com Naive Bayes
  - alpha: Parâmetro de suavização

5.3 Métricas utilizadas

Para a demonstração de eficiência dos algoritmos vamos utilizar três métricas: o f1_score, accuracy_soore e o precision_score. O nosso script base apresenta, para cada algoritmo um relatório com diversas métricas inclusive a nível de classe. Além dessas métricas, o script base ainda apresenta o resultado da função classification_report do scikit-learn. O classification report nos traz ainda a métrica recall_score e cada métrica sendo apresentada nos modos: ‘micro’, ‘macro’ e ‘weighted’. A utilização de diversas métricas é uma das estratégias de tratamento de datasets desbalanceados. Segue um exemplo de relatório gerado pelo script:

```
Training:
OneVsRestClassifier(estimator=LogisticRegression(class_weight='balanced'))
Treino: 0.908s
testes:  0.004s
accuracy:   0.573

Accuracy: 0.10

Micro Precision: 0.54
Micro Recall: 0.61
Micro F1-score: 0.57

Macro Precision: 0.43
Macro Recall: 0.48
Macro F1-score: 0.45

Weighted Precision: 0.54
Weighted Recall: 0.61
Weighted F1-score: 0.57
Relatório de Classificação:
                precision    recall  f1-score   support

        Action       0.56      0.67      0.61       217
     Adventure       0.48      0.60      0.53       146
     Animation       0.28      0.42      0.34        33
        Comedy       0.62      0.71      0.66       331
         Crime       0.46      0.53      0.49       135
   Documentary       0.47      0.38      0.42        24
         Drama       0.69      0.68      0.68       478
        Family       0.42      0.60      0.49        82
       Fantasy       0.39      0.47      0.43        74
       Foreign       0.00      0.00      0.00        11
       History       0.29      0.40      0.34        42
        Horror       0.62      0.59      0.61       116
         Music       0.38      0.47      0.42        30
       Mystery       0.29      0.32      0.30        75
       Romance       0.50      0.63      0.56       188
Science Fiction       0.57      0.62      0.60        98
      TV Movie       0.00      0.00      0.00         2
      Thriller       0.54      0.61      0.58       256
           War       0.44      0.56      0.49        27
       Western       0.55      0.43      0.48        14

     micro avg       0.54      0.61      0.57      2379
     macro avg       0.43      0.48      0.45      2379
  weighted avg       0.54      0.61      0.57      2379
   samples avg       0.55      0.63      0.55      2379

dimensionality: 5000
density: 1.000000
```

5.3.1 Precision Score

O precision score é a razão tp/(tp +fp), onde tp é o número de verdadeiros positivos e fp é o número de falsos positivos. Resumidamente, o precision score mede a habilidade do classificador não classificar como positivo uma amostra que é negativa. O melhor valor é 1 seu pior valor é 0. (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html)

5.3.2 Recall Score

O recall score é a razão tp/(tp +fn), onde tp é o número de verdadeiros positivos e fn é o número de falsos negativos. Resumidamente, o recall score mede a habilidade do classificador encontrar todas as amostras positivas. O melhor valor é 1 seu pior valor é 0.

(https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html)

5.3.3 F1 Score

O F1 score é também conhecido como F-score balanceado ou medida F. O F1 score pode ser interpretado como uma média ponderada da precisão e o recall.O melhor valor é 1 seu pior valor é 0.

A sua fórmula é F1 = 2*(precisao*recall)/(precisao + recall)

(https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

5.3.4 Accuracy Score

O accuracy score em classificações multilabel calcula o conjunto de acurácia em um subconjunto dos dados. O conjunto de labels previstos para uma amostra deve ser exatamente igual ao conjunto de labels dos resultados conhecidos. Essa métrica, aparentemente, não é muito usada. Nas pesquisas realizadas, não foram encontrados muitos exemplos que utilizam o accuracy score. O melhor valor é 1 seu pior valor é 0.

(https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score)

5.4.Trecho base para aplicação dos algoritmos

Vamos ver agora o script base utilizado em todos os modelos de machine learning.

```
1
2  clf  = grid_search_tune.best_estimator_
3  classes=multilabel_binarizer.classes_
4  print('_' * 120)
5  print("Training: ")
6  print(clf)
7  t0 = time()
8  train_time = time() - t0
9  print("Treino: %0.3fs" % train_time)
10
11 t0 = time()
12 pred = clf.predict(test_x)
13 test_time = time() - t0
14 print("testes:  %0.3fs" % test_time)
15
16 score = f1_score(yval, pred, average="micro")
17 precision = precision_score(yval, pred, average='micro')
18 print("accuracy:   %0.3f" % score)
19
20 print('\nAccuracy: {:.2f}\n'.format(accuracy_score(yval, pred)))
21
22 print('Micro Precision: {:.2f}'.format(precision_score(yval, pred, average='micro')))
23 print('Micro Recall: {:.2f}'.format(recall_score(yval, pred, average='micro')))
24 print('Micro F1-score: {:.2f}\n'.format(f1_score(yval, pred, average='micro')))
25
26 print('Macro Precision: {:.2f}'.format(precision_score(yval, pred, average='macro')))
27 print('Macro Recall: {:.2f}'.format(recall_score(yval, pred, average='macro')))
28 print('Macro F1-score: {:.2f}\n'.format(f1_score(yval, pred, average='macro')))
29
30 print('Weighted Precision: {:.2f}'.format(precision_score(yval, pred, average='weighted')))
31 print('Weighted Recall: {:.2f}'.format(recall_score(yval, pred, average='weighted')))
32 print('Weighted F1-score: {:.2f}'.format(f1_score(yval, pred, average='weighted')))
33
34 print("Relatório de Classificação:")
35 print(classification_report(yval, pred,
36                                       target_names=classes))
37
38
39
40
41 clf_descr = 'NaiveBayes-semtratamento-tmdb'#str(clf).split('(')[0]
```

Na linha 2 é setado o estimador com as melhores configurações escolhidas pelo GridSearchCV. Na linha 3 pegamos as classes alvo no multilabel_binarizer. Na linha 12 instruímos o estimador escolhido a fazer previsões no conjunto de dados de teste do dataset. Nas próximas linhas simplesmente imprimimos e armazenamos as métricas da predição realizada.

5.5. Resultados para cada dataset de acordo com o tratamento de dados

TMDB			
Sem Tratamento			
Estimador	**F1_Score**	**Accuracy**	**Precision**
LinearRegression	0.57	0.12	0.59
ExtraTreesClassifier	0.51	0.10	0.56
MultinomialNB	0.46	0.12	0.57
RidgeClassifierCV	0.27	0.03	0.17
RidgeClassifier	0.26	0.03	0.16

TMDB			
Tratamento Parcial			
Estimador	**F1_Score**	**Accuracy**	**Precision**
RidgeClassifier	0.87	0.00	0.05
LinearRegression	0.15	0.04	0.11
ExtraTreesClassifier	0.11	0.03	0.09
RidgeClassifierCV	0.11	0.01	0.06
MultinomialNB	0.02	0.01	0.02

TMDB			
Upsampled			
Estimador	**F1_Score**	**Accuracy**	**Precision**
MultinomialNB	0.57	0.36	0.50
LinearRegression	0.52	0.17	0.37
ExtraTreesClassifier	0.45	0.15	0.32
RidgeClassifierCV	0.15	0.12	0.08
RidgeClassifier	0.01	0.00	0.01

movieLens			
Sem Tratamento			
Estimador	**F1_Score**	**Accuracy**	**Precision**
LinearRegression	0.55	0.14	0.47
ExtraTreesClassifier	0.49	0.12	0.44
MultinomialNB	0.29	0.22	0.60
RidgeClassifierCV	0.22	0.05	0.13
RidgeClassifier	0.20	0.05	0.12

movieLens			
Tratamento Parcial			
Estimador	**F1_Score**	**Accuracy**	**Precision**
LinearRegression	0.31	0.06	0.22
ExtraTreesClassifier	0.27	0.05	0.19
MultinomialNB	0.13	0.07	0.38
RidgeClassifier	0.09	0.01	0.05
RidgeClassifierCV	0.01	0.01	0.05

movieLens			
Upsampled			
Estimador	**F1_Score**	**Accuracy**	**Precision**
LinearRegression	0.51	0.13	0.35
ExtraTreesClassifier	0.43	0.09	0.29
MultinomialNB	0.42	0.27	0.67
RidgeClassifier	0.09	0.01	0.05
RidgeClassifierCV	0.06	0.00	0.03

5.6. Melhores resultados

TMDB		
Top 5 F1 Score		
Estimador	**F1_Score**	**Tratamento**
RidgeClassifier	0.87	Tratamento Parcial
MultinomialNB	0.57	Sem Tratamento
LinearRegression	0.57	Upsampled
LinearRegression	0.52	Upsampled
ExtraTreesClassifier	0.51	Sem Tratamento

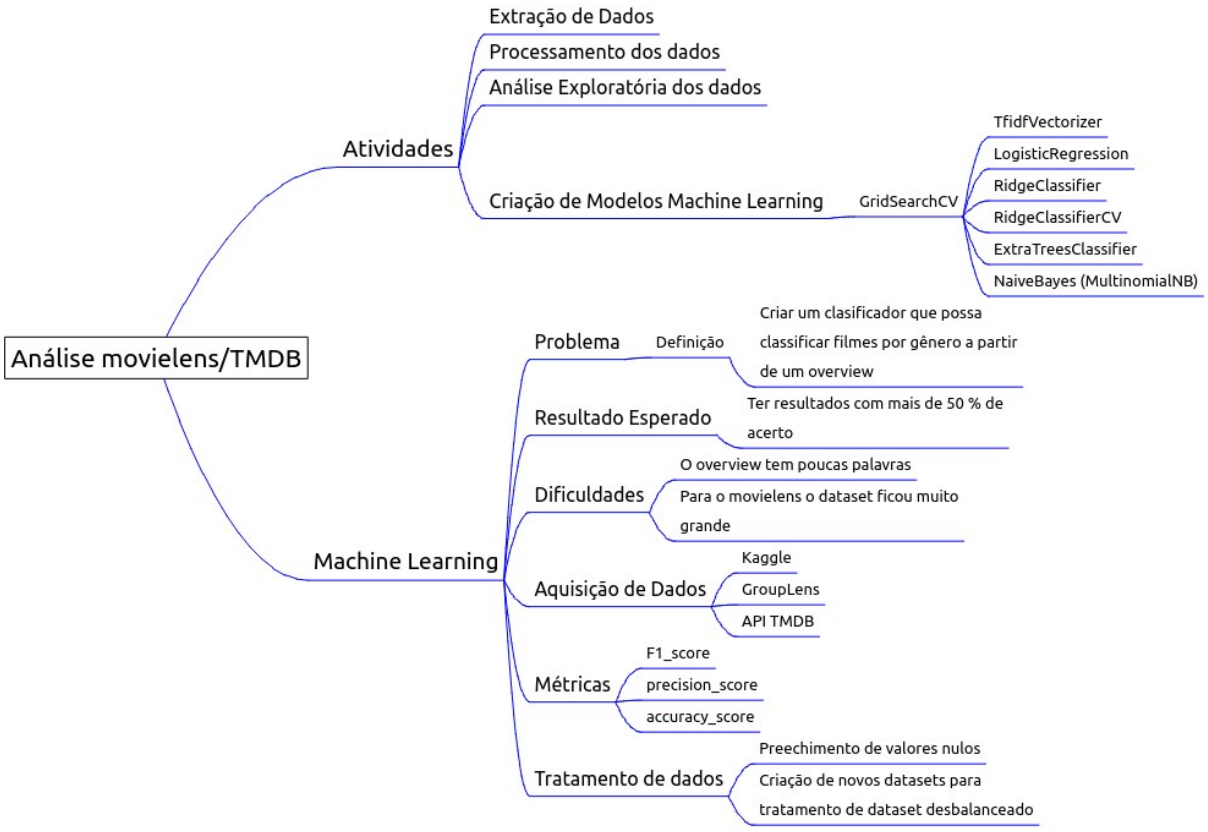
TMDB		
Top 5 Precision		
Estimador	**Precision**	**Tratamento**
LinearRegression	0.59	Sem Tratamento
MultinomialNB	0.57	Sem Tratamento
ExtraTreesClassifier	0.56	Sem Tratamento
MultinomialNB	0.50	Upsampled
LinearRegression	0.37	Upsampled

MovieLens		
Top 5 F1 Score		
Estimador	**F1_Score**	**Tratamento**
LinearRegression	0.55	Sem Tratamento
LinearRegression	0.51	Upsampled
ExtraTreesClassifier	0.49	Sem Tratamento
ExtraTreesClassifier	0.43	Upsampled
MultinomialNB	0.42	Upsampled

MovieLens		
Top 5 Precision Score		
Estimador	**Precision**	**Tratamento**
MultinomialNB	0.67	Upsampled
MultinomialNB	0.60	Sem Tratamento
LinearRegression	0.47	Sem Tratamento
ExtraTreesClassifier	0.44	Sem Tratamento
MultinomialNB	0.38	Tratamento Parcial

6. Apresentação dos Resultados

Vamos ver um mapa mental das atividades desenvolvidas neste trabalho

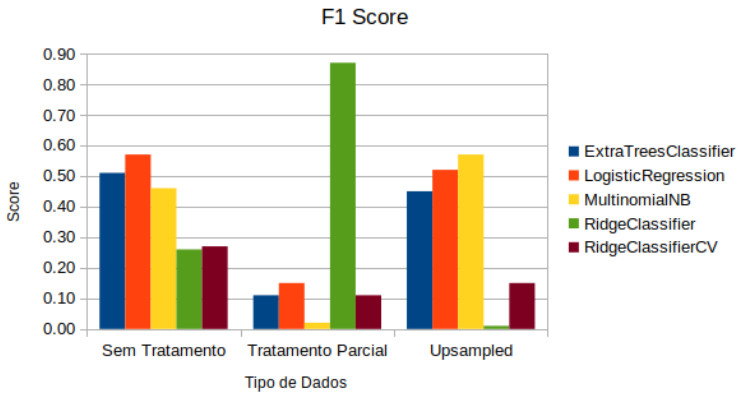


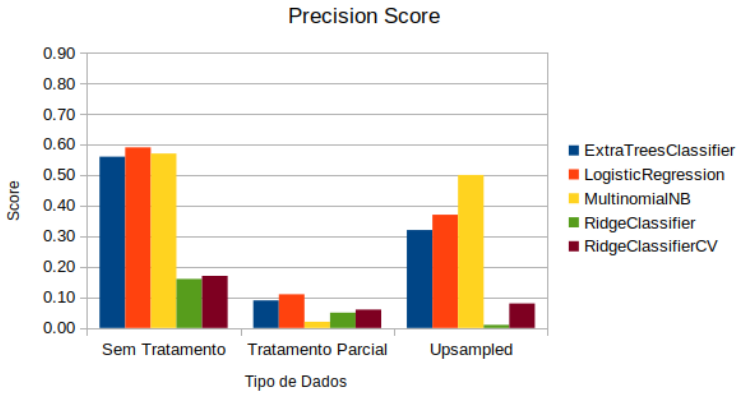
Na imagem acima podemos ver o resumo das atividades desenvolvidas neste trabalho. O trabalho começou com a aquisição de dados de diversas fontes, pegamos datasets prontos no kaggle e no grouplens. Também montamos um dataset a partir do dataset do grouplens e recuperando informações através da API do TMDB.

Na análise exploratória, verificamos o comportamento das médias e notas atribuídas aos filmes dos datasets. Ainda na análise exploratória de dados, conseguimos extrair vários insigths a partir dos dados, pudemos ver quais o gêneros de filmes mais comuns, nuvens das palavras chaves mais utilizadas, nuvens dos gêneros mais utilizados, qual o maior lucro e maior prejuízo obtidos por um filme, como a duração médias dos filmes variaram ao longo dos anos, ainda vimos se existe algum correlacionamento entre o retorno financeiro com o orçamento e/ou duração do filme, entre outros.

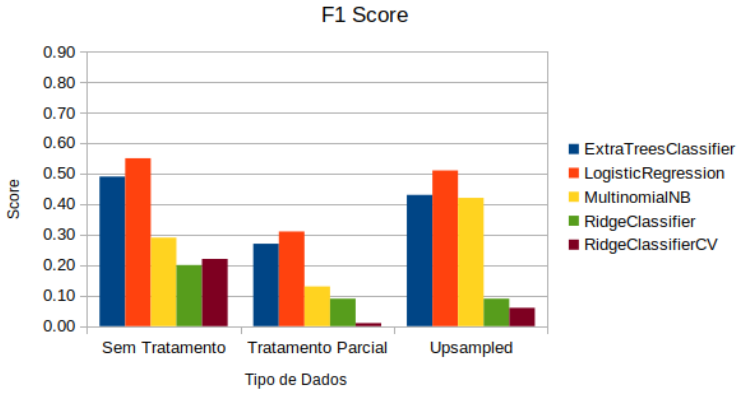
Na parte de criação dos modelos de aprendizagem de máquina, testamos vários algoritmos, com o uso de cross-validation. Conseguimos atingir o objetivo de um modelo de aprendizagem de máquina conseguir tem uma métrica de avaliação maior que 0.5, já que na internet raramente conseguimos ver algum trabalho conseguir atingir essa meta. Também trabalhamos com o tratamento de datasets com classes desbalanceadas, utilizamos algumas técnicas sugeridas, mas pudemos ver nos resultados que os estimadores fornecidos pelo scikit-learn são bem eficientes no tratamento de classes desbalanceadas quando informamos que o peso das classes deve ser balanceado. Pudemos perceber que os melhores resultados foram obtidos quando os dados não tiveram nenhum tratamento sobre as classes desbalanceadas.

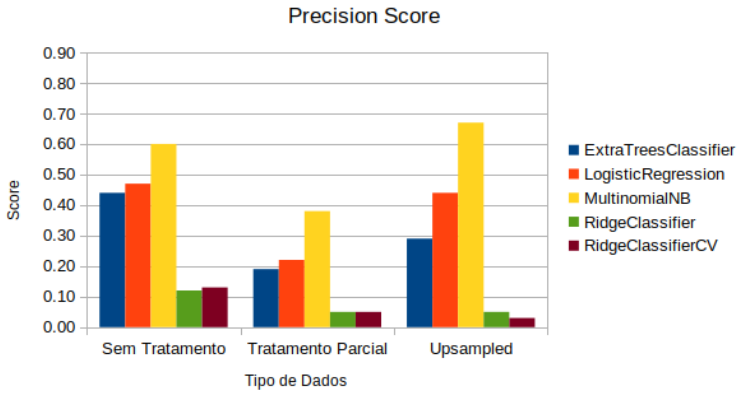
Resultados TMDB:





Resultados movielens:





De acordo com os gráficos acima podemos tirar algumas conclusões:

- Fazer simplesmente a separação dos gêneros não ajudou na performance dos algoritmos de machine learning;

- Tentar fazer o tratamento de classes desbalanceadas para o nosso caso aqui não alterou muito os resultados, além de mudar o tipo de problema de um conjunto de classes multilabel para classes simples;

- Alguns algoritmos como o Multinomial e o ExtraTreesClassfier diminuiu a performance quando aumentou a quantidade de registros, talvez por causa de um overfit;

- Quando vamos trabalhar com machine learning temos que testar diversos algoritmos e diversos parâmetros para tentar achar o melhor conjunto de parâmetros

7. Links

Link para o vídeo: https://www.youtube.com/watch?v=hKll2Vtf2ac

Link para o repositório: https://github.com/igorwc/tcc-puc-minas

REFERÊNCIAS

SCHWARTZ, Barry. **The paradox of choice**. Site: https://www.ted.com/talks/barry_schwartz_the_paradox_of_choice?language=pt-br#t-152755, acessado em 20/07/2020

WALLACE, Frankie. **How Data Science is used within the film Industry**. Site: https://www.kdnuggets.com/2019/07/data-science-film-industry.html, acessado em 07/09/2020

WINTER, Gunnar De. **Artificial Inteligence Goes to Hollywood (and Infiltrates the Movie Industry)**. Site: https://medium.com/predict/artificial-intelligence-goes-to-hollywood-and-infiltrates-the-movie-industry-1c791391b5ba, acessado em 07/09/2020

JOSHI, Naveen. **Can AI automate the film industry.** Site: https://www.allerin.com/blog/can-ai-automate-the-film-industry, acessado em 07/09/2020.

HARPER, F. Maxwell, KONSTAN, Joseph A. **The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS)** 5, 4: 19:1–19:19. 2015. https://doi.org/10.1145/2827872

GLEN, Stephanie. **T Test (Student's T-Test): Definition and Examples.** Site: https://www.statisticshowto.com/probability-and-statistics/t-test/, acessado em 07/09/2020

Brownlee, Jason. **8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset.** Site: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/, acessado em 07/09/2020

Brownlee, Jason. **8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset.** Site: https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/, acessado em 07/09/2020

APÊNDICE

Programação/Scripts

Script de extração de dados da API do TMDB

```
#!/usr/bin/env python
# coding: utf-8
# coding: utf-8
import os
import json
import requests
import pandas as pd
import os.path
import time
import numpy as np

api_token = os.getenv('TMDB_KEY')
api_url_base = 'https://api.themoviedb.org/3/'


def getMovie(tmdbId):
    headers = {'Content-Type': 'application/json' }
    api_url =  '{0}movie/{1}?api_key={2}&language=en-US'.format(api_url_base,tmdbId,api_token)
    response = requests.get(api_url, headers=headers)

    if response.status_code == 200:
        return json.loads(response.content.decode('utf-8'))
    else:
        return None
movies = pd.read_csv('../../../dados/movielens/ml-25m/movies.csv')
links = pd.read_csv('../../../dados/movielens/ml-25m/links.csv')
# Aqui tive que fazer um slice dos dados para não bloquear acesso à API do TMDB
temp = links[55000:]
i = 0
for index, row in temp.iterrows():
    i = i +1
    if(i%100 == 0):
        print('filmes: '+str(i))
        time.sleep(10)
    try:
        movieId = str(int(row['tmdbId']))

        if (not os.path.exists('../../../dados/tmdb/'+movieId+'.json')):
            x = getMovie(movieId)

            with open('../../../dados/tmdb/'+movieId+'.json', 'w') as outfile:
                json.dump(x, outfile, indent=4)
    except:
        continue
links.dropna(inplace=True)

links.dropna(inplace=True)
dados = []
for index, row in links.iterrows():
    movieId = str(int(row['tmdbId']))
    if (os.path.isfile('../../../dados/tmdb/'+movieId+'.json')):

        with open('../../../dados/tmdb/'+movieId+'.json') as json_file:
            print(movieId)
            data = json.load(json_file)
            if(data  is None or type( data) == None or (not 'overview' in data)):
                dados.append((row['movieId'], ''))
            else:
                dados.append((row['movieId'], data['overview']))
dados = dict(dados)
overview_df = pd.DataFrame.from_dict(dados, orient='index')
overview_df.to_csv('../../../dados/tmdb/overview.csv')
overview_df.reset_index()
overview_df['index1'] = overview_df.index
overview_df.columns = ['overview','movieId']
filmes_com_overview = pd.merge(movies, overview_df, on="movieId", how="right")
filmes_com_overview['overview'].replace('', np.nan, inplace=True)
filmes_com_overview.dropna(subset=['overview'], inplace=True)
filmes_com_overview.to_csv('../../../dados/movielens/filmes_com_overview.csv')
```

Gráficos

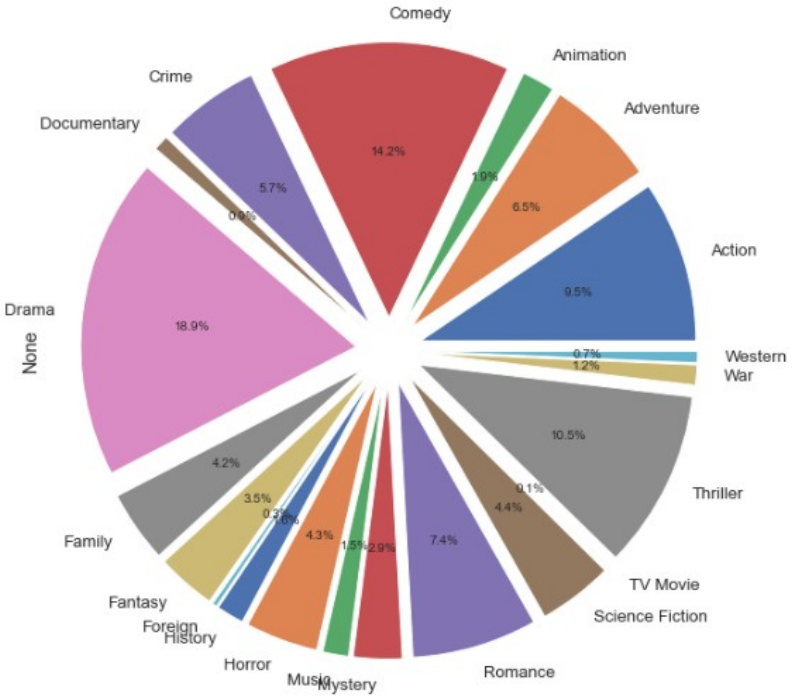


Gráfico de pizza dos gêneros dos filmes no TMDB