UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA

Metodologia de Avaliação da disciplina Inteligência Artificial - Segunda Nota

Validação de Dados Sob Perspectiva dos Algoritmos Descritos em Machine Learning

SILVA A. S.
2016022764

Disciplina: Inteligência Artificial

Prof.A: Thaís Gaudêncio

JOÃO PESSOA, 9 de março de 2020

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

A disciplina “Inteligência Artificial” da Universidade Federal da Paraíba aborda conceitos de *Machine Learning* e *Deep Learnig*, com objetivo de transpor ao discente a capacidade de compreender problemas do mundo real, tornando possível deslumbrar a contribuição que a análise de dados pode trazer para a tecnologia moderna e avanço da sociedade.

O conteúdo abordado neste trabalho corresponde a ementa de avaliação da disciplina, de modo a por em pratica a contextualização teórica vista em sala de aula.
Para composição da avaliação, foram determinados a resolução de três problemas de analise de dados relacionados ao cotidiano moderno, sob a respectiva da implementação de algoritmos de aprendizagem de maquina.

O primeiro problema disposto trata da implementação de um algoritmo de classificação supervisionada[1] para um problema determinado na literatura como “problema de classificação *multi-label*[2], a partir dos conceitos vistos sobre aprendizagem de máquina e seus respectivos algoritmos para classificação, neste caso, os objeto de estudos seriam uma diversidade de músicas classificadas com base nas emoções humanas. O desafio se tornou mais complexo quando o problema dispunha de dados que requeriam uma análise para a possibilidade de que uma musica poderia pertencer a mais de uma classe de emoções ao mesmo tempo.

No segundo problema, lidamos com o conceito de regressão[3], também de modo supervisionado, o problema visou implementar métodos de aprendizagem de máquina para identificar o que apresentaria melhores resultados na previsão do volume de tráfego de um determinado local.

O terceiro problema, baseou-se na perspectiva de utilização dos algoritmos de agrupamento[4] para inferir sobre uma base de dados que descreve o comportamento dos usuários nas redes sociais, com intuito de agrupar os semelhantes por tipo de publicação.

Nas sessões seguintes serão apresentados os métodos de avaliação dos problemas, as formas de tratamento dos dados, o pré-processamento ante a análise pelos algoritmos de aprendizagem, e por fim, as estatísticas de resultados gerados bem como a compreensão do pesquisador ante os resultados, para cada um dos problemas em ordem ascendente.

---

1 Referências [1]
2 Referências [2]
3 Referências [3]
4 Referências [4]

2. Primeiro Problema - Classificação

2.1 Metodologia

A base de dados "*emotions*" é composta por 593 (quinhentos e noventa e três) instâncias que representam características de músicas, e foi dividida em 78 (setenta e oito) colunas, onde as últimas 6 (seis) colunas, correspondem a classificação binária do pertencimento ou não pertencimento de cada objeto à referida classe de emoções descritas pelo *dataset. A*s 72 (setenta e duas) colunas restantes são referentes aos atributos, que descrevem os objetos e representam medidas individuais desconhecidas, de cada uma das músicas documentadas.

O objetivo da análise é encontrar uma correlação entre os atributos das músicas e as classes de emoções, de modo a conseguir prever, com base em atributos de entrada, a qual classe de emoções a música em questão pertenceria.

Neste problema, abstraiu-se a compreensão sobre o significado do conteúdo dos atributos, visto que descritos numericamente, não seria possível para o pesquisador compreender sua natureza com base apenas na leitura do *dataset*.

2.2 Pré-processamento dos dados

2.2.1 Análise exploratória

A base de dados "*emotions.arff*" foi importada para o *Jupyter Notebook* utilizando *Pandas*, a biblioteca da linguagem Python , e na imagem à seguir está descrito o conjunto de código responsável por representar as dimensões e a representação do cabeçalho contendo as 5 (cinco) primeiras instâncias da base .

```
In [147]: print("Number of rows in data =",df.shape[0])
          print("Number of columns in data =",df.shape[1])
          print("\n")

          printmd("**Sample data:**")
          df.head()
```

```
Number of rows in data = 593
Number of columns in data = 78
```

Sample data:

Out[147]:

	Mean_Acc1298_Mean_Mem40_Centroid	Mean_Acc1298_Mean_Mem40_Rolloff	Mean_Acc1298_Mean_Mem40_Flux	Mean_Acc1
0	0.034741	0.089665	0.091225	
1	0.081374	0.272747	0.085733	
2	0.110545	0.273567	0.084410	
3	0.042481	0.199281	0.093447	
4	0.074550	0.140880	0.079789	

5 rows × 78 columns

Figura 1 - Base de dados *'emotions'*

2.2.2 Dados discrepantes

O próximo passo do pré-processamento foi a verificação valores faltantes, e remoção dos valores duplicados:

```
In [148]: df.isnull().values.any()

Out[148]: False

In [149]: df.drop_duplicates().values

Out[149]: array([[0.034741, 0.089665, 0.091225, ..., b'0', b'0', b'0'],
                 [0.081374, 0.272747, 0.085733, ..., b'0', b'0', b'1'],
                 [0.110545, 0.273567, 0.08441, ..., b'0', b'0', b'1'],
                 ...,
                 [0.035169, 0.065403, 0.075227, ..., b'1', b'1', b'0'],
                 [0.054276, 0.238158, 0.095935, ..., b'0', b'0', b'0'],
                 [0.073194, 0.140733, 0.080545, ..., b'0', b'0', b'0']],
                dtype=object)
```

Figura 2 - Valores faltantes e duplicados

2.2.3 Explorando os *labels* de classificação

A seguir analisaremos as colunas correspondentes às categorias de emoções descritas no *dataset*: ***'amazed-suprised', 'happy-pleased', 'relaxing-calm', 'quiet-still', 'sad-lonely', 'angry-aggresive'***

```
categories = list(df.columns.values)
categories = categories[72:]
print(categories)
```

```
['amazed-suprised', 'happy-pleased', 'relaxing-calm', 'quiet-still', 'sad-lonely', 'angry-a
ggresive']
```

amazed-suprised	happy-pleased	relaxing-calm	quiet-still	sad-lonely	angry-aggresive
b'0'	b'1'	b'1'	b'0'	b'0'	b'0'
b'1'	b'0'	b'0'	b'0'	b'0'	b'1'
b'0'	b'1'	b'0'	b'0'	b'0'	b'1'
b'0'	b'0'	b'1'	b'0'	b'0'	b'0'
b'0'	b'0'	b'0'	b'1'	b'0'	b'0'

Figura 3 - *Labels* de classificação

A partir dessa análise é possível observar que os dados estão representados com uma codificação de *string* binária, o que sugeriu a necessidade de converter os valores para uma medida numérica, de 1 (uns) e 0 (zeros) . Depois desse tratamento, foi possível contar os atributos que pertencem a cada uma das categorias, lembrando que, alguns podem pertencer a mais de uma categoria ao mesmo tempo.

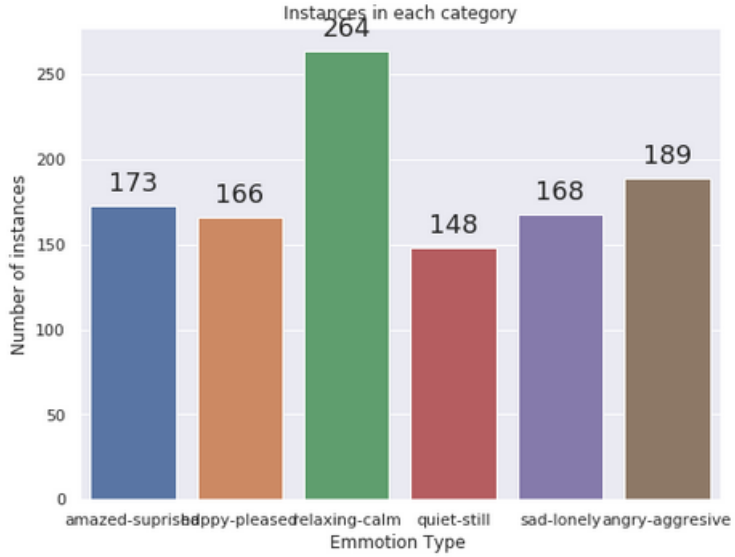


Figura 4 - Número de instâncias por categoria

Seguindo esta linha de raciocínio, o próximo passo é a verificação de quantos atributos pertencem a mais de uma classe ao mesmo tempo:

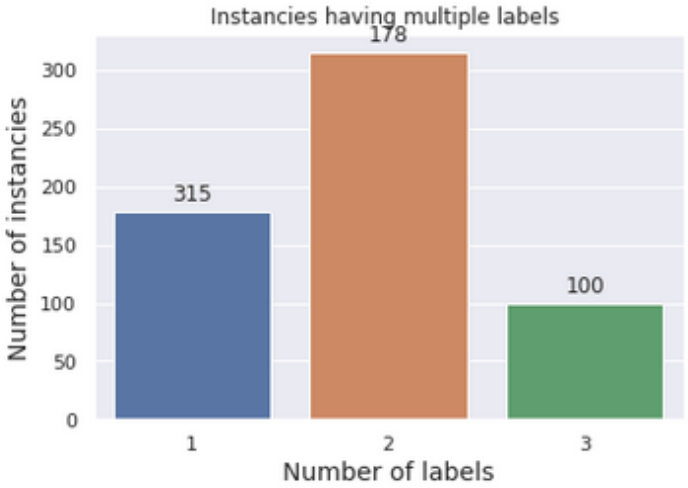


Figura 5 - Número de instâncias *multi-label*

O gráfico descreve que existem 315 (trezentos e quinze) atributos que pertencem a apenas uma classe, 178 (cento e setenta e oito) pertencem a duas classes, e 100 (cem) atributos pertencem à 3 (três) classes ao mesmo tempo, sendo o último o maior numero de *labels* de um atributo na base, apesar de existirem 6 *labels* de classificação.

Partindo para a exploração da existência de *outliers* (atributos com valores muito divergentes da media dos valores gerais), um gráfico de *boxplot e* um gráfico de barras, foram gerados sob os valores do *dataset* para melhor visualização de possíveis *outliers.*

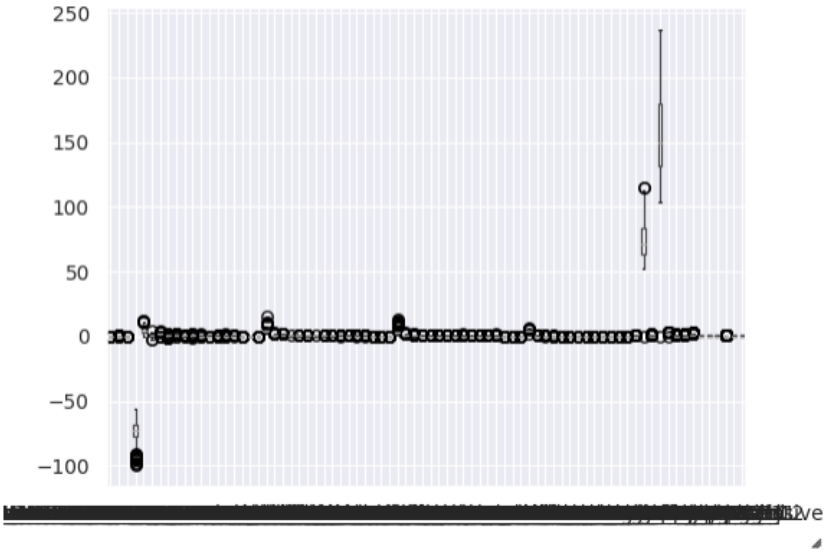


Figura 6 - Gráfico de BoxPlot

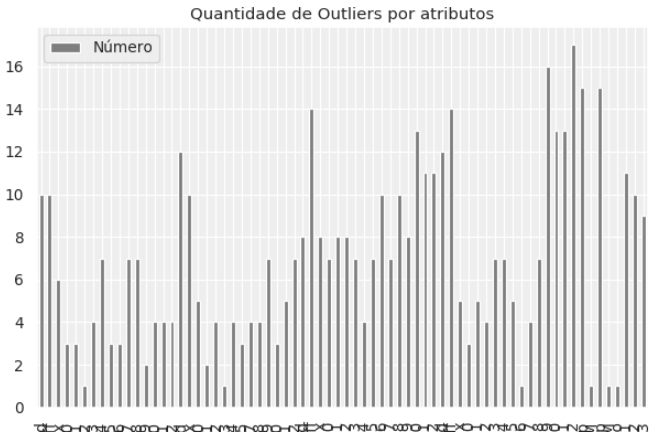
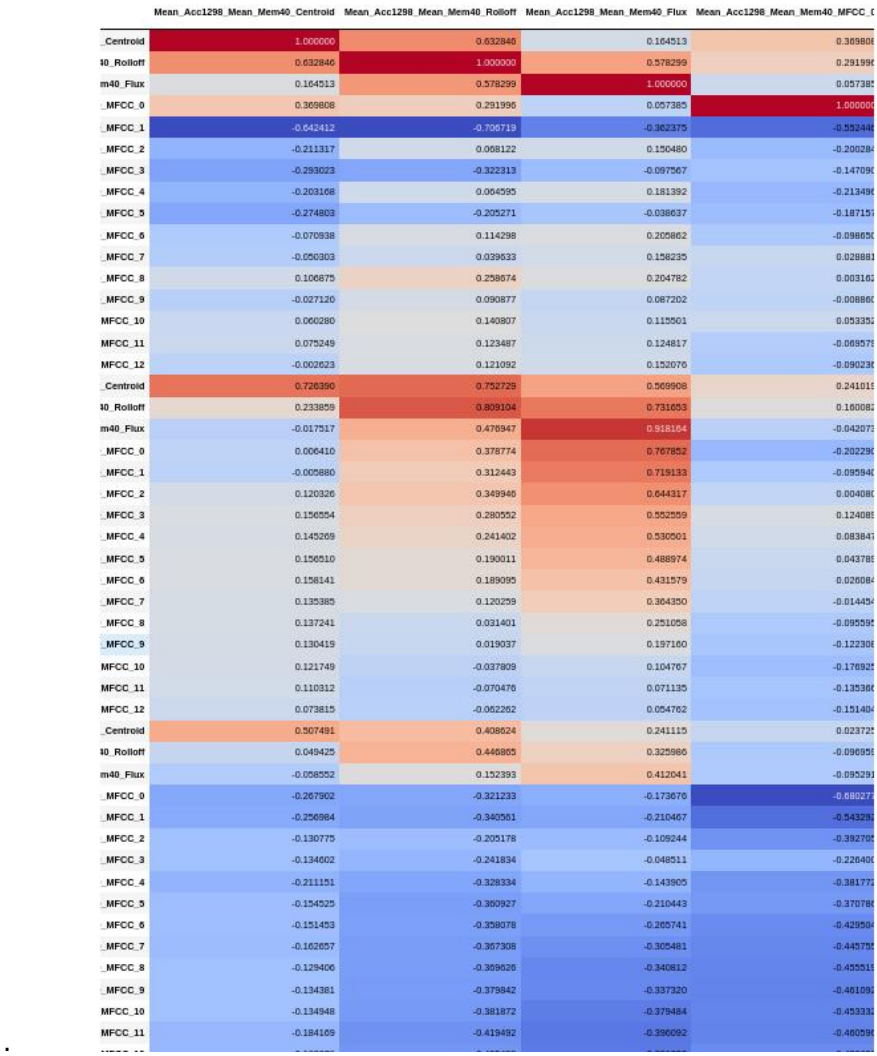


Figura 7 - Gráfico de barras

De modo que, apesar da aparente existência de *outliers*, por desconhecer a natureza dos atributos, não podemos determinar com certeza de que os dados realmente são *outliers*, sendo assim, optou-se por manter os dados originais e não removê-los, do *dataset*.

2.2.4 Correlação dos atributos

Entende-se como correlação qualquer relação estatística dentro de uma ampla classe de relações que envolva dependência entre duas variáveis.
Diante disto, uma matriz de correlação entre os atributos e as categorias foi gerada, no padrão “*coolrwarm*”, que destaca as cores mais próximas do vermelho como os maiores valores de correlação. Contudo, a matriz tem uma grande dimensionalidade devido às existentes 78 (setenta e oito) colunas de atributos, dessa maneira, a seguir está ilustrada uma pequena fração da matriz, que pode ser conferida integralmente no arquivo de *notebook* disponibilizado junto com este relatório.



	Mean_Acc1298_Mean_Mem40_Centroid	Mean_Acc1298_Mean_Mem40_Rolloff	Mean_Acc1298_Mean_Mem40_Flux	Mean_Acc1298_Mean_Mem40_MFCC_0
_Centroid	1.000000	0.632846	0.164513	0.369808
40_Rolloff	0.632846	1.000000	0.578299	0.291996
m40_Flux	0.164513	0.578299	1.000000	0.057385
_MFCC_0	0.369808	0.291996	0.057385	1.000000
_MFCC_1	-0.642412	-0.706719	-0.362375	-0.552448
_MFCC_2	-0.211317	0.068122	0.150480	-0.200284
_MFCC_3	-0.293023	-0.322313	-0.097567	-0.147096
_MFCC_4	-0.203168	0.064595	0.181392	-0.213496
_MFCC_5	-0.274803	-0.205271	-0.038637	-0.187157
_MFCC_6	-0.070938	0.114298	0.205862	-0.098650
_MFCC_7	-0.050303	0.039633	0.158235	0.028881
_MFCC_8	0.106875	0.258574	0.204782	0.003161
_MFCC_9	-0.027120	0.090877	0.087202	-0.008860
MFCC_10	0.060280	0.140807	0.115501	0.053352
MFCC_11	0.075249	0.123487	0.124817	-0.068575
MFCC_12	-0.002623	0.121092	0.152076	-0.090230
_Centroid	0.726390	0.752729	0.569908	0.241015
40_Rolloff	0.233859	0.809104	0.731653	0.160082
m40_Flux	-0.017517	0.476947	0.918164	-0.042073
_MFCC_0	0.006410	0.378774	0.767852	-0.202290
_MFCC_1	-0.005880	0.312443	0.719133	-0.095940
_MFCC_2	0.120326	0.349946	0.644317	0.004080
_MFCC_3	0.156554	0.280552	0.552559	0.124085
_MFCC_4	0.145269	0.241402	0.530501	0.083847
_MFCC_5	0.156510	0.190011	0.488974	0.043785
_MFCC_6	0.158141	0.189095	0.431579	0.026084
_MFCC_7	0.135385	0.120259	0.364350	-0.014454
_MFCC_8	0.137241	0.031401	0.251058	-0.095595
_MFCC_9	0.130419	0.019037	0.197160	-0.122308
MFCC_10	0.121749	-0.037809	0.104767	-0.176925
MFCC_11	0.110312	-0.070476	0.071135	-0.135366
MFCC_12	0.073815	-0.062262	0.054762	-0.151404
_Centroid	0.507491	0.408624	0.241115	0.023725
40_Rolloff	0.049425	0.446865	0.325986	-0.096951
m40_Flux	-0.058552	0.152393	0.412041	-0.095291
_MFCC_0	-0.267902	-0.321233	-0.173676	-0.680273
_MFCC_1	-0.256984	-0.340561	-0.210467	-0.543291
_MFCC_2	-0.130775	-0.205178	-0.109244	-0.392705
_MFCC_3	-0.134602	-0.241834	-0.048511	-0.226400
_MFCC_4	-0.211151	-0.328334	-0.143905	-0.381771
_MFCC_5	-0.154525	-0.360927	-0.210443	-0.370780
_MFCC_6	-0.151453	-0.358978	-0.265741	-0.429504
_MFCC_7	-0.162657	-0.367308	-0.305481	-0.445751
_MFCC_8	-0.129406	-0.369526	-0.340812	-0.455519
_MFCC_9	-0.134381	-0.379842	-0.337320	-0.461092
MFCC_10	-0.134948	-0.381872	-0.379484	-0.453331
MFCC_11	-0.184169	-0.419492	-0.396092	-0.460596

Figura 8 - Correlação dos dados

2.2.5 Balanceamento das classes

Foi verificado pela execução ilustrada na próxima figura, que os dados referentes aos *labels* de classificação possuíam uma discrepância em relação às quantidades de atributos pertencendo a cada categoria, como a classificação é feita de maneira binária, o numero de atributos pertencentes (um) ou não pertencentes (zero) tinha uma diferença ampla.
Com intuito de minimizar o impacto dessa diferença no algoritmo de aprendizagem, optou-se pela realização do "*downsample*", método de balanceamento onde uma parcela das instancias com maior número é removida da base de dados na intenção de obter o mesmo valor de instancias correspondentes ao pertencimento ou não da classe na coluna que a descreve.

A seguir a quantidade de atributos por valores de classificação em cada categoria:

```
print(data['amazed-suprised'].value_counts(),"\n",
data['happy-pleased'].value_counts(),"\n",
data['relaxing-calm'].value_counts(),"\n",
data['quiet-still'].value_counts(),"\n",
data['sad-lonely'].value_counts(),"\n",
data['angry-aggresive'].value_counts())
```

```
0    420
1    173
Name: amazed-suprised, dtype: int64
 0    427
1    166
Name: happy-pleased, dtype: int64
 0    329
1    264
Name: relaxing-calm, dtype: int64
 0    445
1    148
Name: quiet-still, dtype: int64
 0    425
1    168
Name: sad-lonely, dtype: int64
 0    404
1    189
Name: angry-aggresive, dtype: int64
```

Figura 9 - Quantidade de atributos por classificação

Abaixo o resultado das quantidades após a realização do *downsample:*

```
: new_data = downsample(data, 'amazed-suprised', 0, 1 , 173)

1    173
0    173
Name: amazed-suprised, dtype: int64

: new_data = downsample(data, 'happy-pleased', 0, 1 , 166)

1    166
0    166
Name: happy-pleased, dtype: int64

: new_data = downsample(data, 'relaxing-calm', 0, 1 , 264)

1    264
0    264
Name: relaxing-calm, dtype: int64

: new_data = downsample(data, 'quiet-still', 0, 1 , 148)

1    148
0    148
Name: quiet-still, dtype: int64

: new_data = downsample(data, 'sad-lonely', 0, 1 , 168)

1    168
0    168
Name: sad-lonely, dtype: int64

: new_data = downsample(data, 'angry-aggresive', 0, 1 , 189)

1    189
0    189
Name: angry-aggresive, dtype: int64

: %matplotlib notebook
new_data['angry-aggresive'].plot(kind = 'hist')
```

Figura 10 - Atributos após balanceamento

E um gráfico para ilustrar o balanceamento de uma das categorias como exemplo da execução:

```
%matplotlib notebook
new_data['angry-aggresive'].plot(kind = 'hist')
```

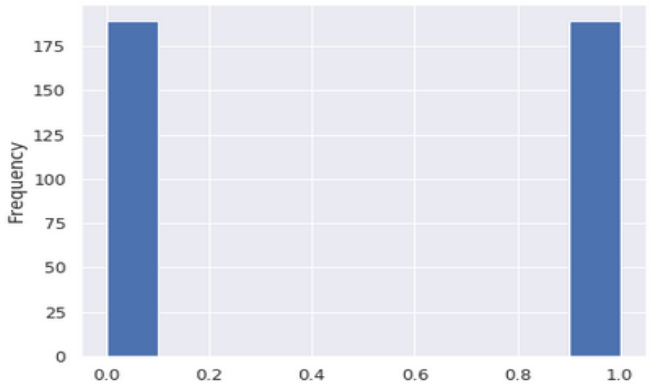


Figura 11 - Gráfico do balanceamento

Após a remoção dos atributos a base final ficou com 378 (trezentos e setenta e oito) instâncias:

```
In [172]: new_data.shape
Out[172]: (378, 78)

In [173]: new_data.head()
Out[173]:
```

	Mean_Acc1298_Mean_Mem40_Centroid	Mean_Acc1298_Mean_Mem40_Rolloff	Mean_Acc1298_Mean_Mem40_Flux	Mean_Acc1298_Mean_Mem40_MFCC_0	M
275	0.061391	0.134275	0.085648	-69.242042	
73	0.088223	0.156644	0.082247	-74.329536	
191	0.061903	0.121791	0.077300	-81.929769	
306	0.089745	0.265649	0.076258	-64.273652	
478	0.035760	0.053166	0.077110	-78.220551	

5 rows × 78 columns

Figura 12 - Instâncias após balanceamento

2.2.6 Normalização

Normalização dos dados é uma técnica de pré-processamento que visa minimizar o viés de importância que o algoritmo de aprendizagem pode dar a uma determinada instância, quando os valores são diferentes entre si. Dessa maneira, o método visa substituir todos os valores para uma escala compreendida entre 0 (zero) e 1 (um), positivo ou negativo, e seu resultado está ilustrado a seguir:

```
In :
from sklearn import preprocessing

normalized = new_data

normal = preprocessing.Normalizer()
normalized = pd.DataFrame(normal.fit_transform(new_data), columns = new_data.columns.values)
normalized['angry-aggresive'] = new_data['angry-aggresive'].values
normalized.head()
```

```
Out:
```

	Mean_Acc1298_Mean_Mem40_Centroid	Mean_Acc1298_Mean_Mem40_Rolloff	Mean_Acc1298_Mean_Mem40_Flux	Mean_Acc1298_Mean_Mem40_MFCC_0	Mea
0	0.000434	0.000949	0.000605	-0.489335	
1	0.000526	0.000935	0.000491	-0.443562	
2	0.000361	0.000711	0.000451	-0.478378	
3	0.000628	0.001859	0.000534	-0.449770	
4	0.000213	0.000317	0.000460	-0.466956	

5 rows × 78 columns

Figura 13 - Resultado da Normalização

2.3 Treinamento e testes

2.3.1 Divisão da base

Para treinar um algoritmo de classificação, e ter possibilidade de testar os resultados posteriormente, a base de dados deve ser dividida em duas parcelas. Nessa abordagem consideramos manter 80% da base original para realizar o treinamento, e 20% da base para realização dos testes de classificação. Os atributos para compor o treinamento e o teste foram determinados de maneira aleatória usando as bibliotecas do *python3*, e a execução esta ilustrada pelo código à seguir:

```
from sklearn.model_selection import train_test_split


X = normalized.iloc[:, :-6]
y = normalized.iloc[:, [72,73,74,75,76,77]]

x_train, x_test, y_train, y_test = train_test_split(X,y, random_state=4, test_size=0.2, shuffle=True)
```

```
print(x_train.shape)
print(x_test.shape)
```

```
(302, 72)
(76, 72)
```

Figura 14 - Divisão Treinamento e Testes

Onde “x” corresponde às colunas que representam os atributos das músicas, e o “y” aos *labels* de classificação, isolados a partir da coluna 72 (setenta e dois).

2.3.2 Testes

Num problema de classificação *multi-label*, a literatura dispõe de diversas técnicas de abordagens definidas como *Problem Transformation Method* e *Adapted Algorithm Method,* onde se visa desde transformar um problema *multi-label* em *single-label* (um *label* apenas) até adaptar algoritmos já conhecidos como o *kNN* para a resolução da classificação.

Diante disto, a neste trabalho foi decidido passar por algumas dessas técnicas visando determinar qual se adaptaria melhor ao problema proposto.

2.3.2.1 *One vs Rest*

A primeira classificação utilizou o algoritmo *One vs Rest*[5] para tratamento do problema *multi-label,* que visa treinar um único classificador por classe, e a Regressão Logística[6] como método de aprendizagem de máquina para classificação.
Entende-se por regressão logística a "*técnica estatística, que tem como objetivo produzir, um modelo que permita a predição de valores tomados por uma variável categórica, frequentemente binária, a partir de uma série de variáveis explicativas contínuas e/ou binárias"* (Wikipedia, 2020).

Medidas da acurácia do teste foram determinadas para avaliar o desempenho.

```
LogReg_pipeline = Pipeline([
                ('clf', OneVsRestClassifier(LogisticRegression(solver='sag'), n_jobs=-1)),
            ])

for category in categories:
    printmd('**Processing {} categorie**'.format(category))

    LogReg_pipeline.fit(x_train, train[category])

    prediction = LogReg_pipeline.predict(x_test)
    print('Test accuracy is {}'.format(accuracy_score(test[category], prediction)))
    print("\n")
```

Processing amazed-suprised categorie

Test accuracy is 0.4868421052631579

Processing happy-pleased categorie

Test accuracy is 0.6973684210526315

Processing relaxing-calm categorie

Test accuracy is 0.631578947368421

Processing quiet-still categorie

Test accuracy is 0.7368421052631579

Processing sad-lonely categorie

Test accuracy is 0.6973684210526315

Processing angry-aggresive categorie

Test accuracy is 0.5131578947368421

Figura 15 - Regressão Logistica One vs Rest

De maneira que, a técnica *One vs Rest*, apesar de popular apresenta problemas de confiabilidade nos valores da classificação binária, mesmo num conjunto de dados balanceado, e a maior acurácia do teste foi para a categoria de ***"quiet-still"*** com 73% de acerto, e ainda, considerando que essa abordagem não é eficiente para classificar as instâncias *multi-label* em classes *multi-label* de forma simultânea, foi decidido adotar outra estratégia de análise.

---

5 Referências [6]

6 Referências [7]

2.3.2.2 Label PowerSet

A segunda abordagem do problema *multi-label* considerou a técnica de *Label Powerset*[7], que se caracteriza por criar novos *labels* de classificação para as instâncias que são *multi-label*, de maneira a considerar as uniões dos *labels* em uma nova categoria, por exemplo: Suponhamos que uma instância pode fazer parte da primeira e da segunda classe ao mesmo tempo, logo, a técnica cria uma nova classe determinada pela união da primeira e da segunda, representada com "*primeira & segunda*" e classifica positivamente o atributo nesse novo *label*.

Para execução da aprendizagem foram escolhidos o algoritmo SVM[8] (System vector machines) e KNN[9] (*K-nearest neighbor)* , o primeiro pela sua reconhecida eficiência em tarefas de classificação, visto que, até a popularidade das redes neurais, era considerado o melhor tipo de algoritmo para essa tarefa por alguns autores; E o segundo pela sua popularidade ante a ser conhecido como um método que não gera exatamente um "modelo" final, mas classifica pela proximidade da distância euclidiana e é considerado um algoritmo de "aprendizagem preguiçosa".

```
classifier = LabelPowerset(svm.SVC())
classifier2 = LabelPowerset(neighbors.KNeighborsClassifier())

classifier.fit(x_train, y_train)
classifier2.fit(x_train, y_train)

predictions = classifier.predict(x_test)
predictions2 = classifier2.predict(x_test)

print("Accuracy1 = ",accuracy_score(y_test,predictions))
print("Accuracy2 = ",accuracy_score(y_test,predictions2))
print("\n")
```

```
Accuracy1 =  0.5
Accuracy2 =  0.39473684210526316


CPU times: user 94.1 ms, sys: 15.4 ms, total: 109 ms
Wall time: 1.72 s
```

Figura 16 - Algoritimos associados ao Label PowerSet

Por hora, apenas considerando a acurácia para medida de desempenho, o algoritmo SVM classificador se mostrou mais eficiente que o KNN classificador, como já era esperado pela contextualização de ambos os métodos, com acerto de 50% na classificação em detrimento de 39% do KNN, que utilizou o valor *default* da biblioteca de $k = 5$ (número de vizinhos mais próximos).

---

7 Referências [8]

8 Referências [9]

9 Referências [10]

2.3.2.3 MLkNN

Por fim, para instanciar a abordagem de adaptação de algoritmos para classificação *multi-label* aplicamos o algoritmo *MLkNN* que adapta a aprendizagem do KNN para um problema *multi-label*. Essa implementação alcançou o resultado de 39% de acurácia no teste para o valor de k = 3 , como pode ser visto abaixo :

```
from skmultilearn.adapt import MLkNN
from scipy.sparse import csr_matrix, lil_matrix
```

```
%%time

classifier_new = MLkNN(k=3)


x_train = lil_matrix(x_train).toarray()
y_train = lil_matrix(y_train).toarray()
x_test = lil_matrix(x_test).toarray()

classifier_new.fit(x_train, y_train)

predictions_new = classifier_new.predict(x_test)

print("Accuracy = ",accuracy_score(y_test,predictions_new))
print("\n")
```

```
Accuracy =  0.39473684210526316


CPU times: user 337 ms, sys: 0 ns, total: 337 ms
Wall time: 800 ms
```

Figura 17 - MLkNN

2.3.3 Avaliação de resultados

Diante do disposto, consideramos que a abordagem mais eficiente foi a utilização da associação de *Label PowerSet* com *SVM* para classificação, seguido pelo algoritmo *KNN*, que apresentou acurácia parecida dentro dessa abordagem e também na teoria de adaptação do algoritmo para o *MLkNN* com os valores de *k* = 5, e *k* = 3 respectivamente.

Abaixo, apresentamos a matriz de confusão por categoria, e medidas de *f1*,sensibilidade, especificidade e acurácia, na execução do *LabelPowerSet* para melhor interpretação dos resultados.

```
Medidas de Desempenho1 svm :
F1: 0.643
Sensibilidade: 0.648
Especificidade: 0.870
Acurácia: 0.500
Média acurácia individual: 0.816

Matriz de confusão para amazed-suprised
[[39 15]
 [ 5 17]]

Matriz de confusão para happy-pleased
[[54  2]
 [19  1]]

Matriz de confusão para relaxing-calm
[[44  0]
 [13 19]]

Matriz de confusão para quiet-still
[[58  1]
 [ 3 14]]

Matriz de confusão para sad-lonely
[[57  2]
 [ 6 11]]

Matriz de confusão para angry-aggresive
[[25 18]
 [ 0 33]]
```

Figura 18 - Desempenho SVM

```
Medidas de Desempenho2 knn :
F1: 0.635
Sensibilidade: 0.612
Especificidade: 0.872
Acurácia: 0.395
Média acurácia individual: 0.796

Matriz de confusão para amazed-suprised
[[45  9]
 [ 9 13]]

Matriz de confusão para happy-pleased
[[46 10]
 [16  4]]

Matriz de confusão para relaxing-calm
[[43  1]
 [15 17]]

Matriz de confusão para quiet-still
[[56  3]
 [ 3 14]]

Matriz de confusão para sad-lonely
[[55  4]
 [ 5 12]]

Matriz de confusão para angry-aggresive
[[31 12]
 [ 6 27]]
```

Figura 19 - Desempenho kNN

Evidentemente, o resultado das abordagens escolhidas não obteve um resultado final amplamente satisfatório, as suposições para esta causalidade estão na complexidade de tratar instâncias *multi-label* quando a correlação entre os atributos é baixa, a incerteza em relação à existência de *outliers* e a adaptação dos algoritmos escolhidos a esta base especificamente.

De modo que, para contribuições futuras outras métricas de avaliação para esse problema possam vir a ser consideradas a fim de alcançar resultados mais satisfatórios no que diz respeito as taxas de erro do classificador.

3. Segundo Problema - Regressão

3.1 Metodologia

"Metro_Interstate_Traffic_Volume" é um *dataset* que descreve informações relevantes ao transito sobre variáveis do cotidiano e as relaciona com o volume previsto de tráfego urbano.
Composta por 48 204 (quarenta e oito mil, duzentos e quatro) instâncias em 9 (nove) colunas, onde a última coluna corresponde ao volume do tráfego, e é o atributo que objetivamos prever nesta análise. O volume do trafego está representado por um número inteiro, variando em uma escala de 0 (zero) à 7280 (sete mil duzentos e oitenta).
O objetivo da análise é encontrar um modelo preditivo regressor supervisionado[10] para determinar o volume de tráfego baseado nas instâncias de entrada apresentadas.

3.2 Pré-processamento dos dados

3.2.1 Análise exploratória

A base de dados "*Metro_Interstate_Traffic_Volume.csv*" foi Importada ao *Jupyer notebook*, também utilizando a biblioteca *Pandas*, e na imagem à seguir está descrito o conjunto de código responsável por representar as dimensões e a representação do cabeçalho contendo as 5 (cinco) primeiras instâncias da base .

```
df = pd.read_csv("Metro_Interstate_Traffic_Volume.csv")
```

```
df.head()
```

temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_volume
288.28	0.0	0.0	40	Clouds	scattered clouds	2012-10-02 09:00:00	5545
289.36	0.0	0.0	75	Clouds	broken clouds	2012-10-02 10:00:00	4516
289.58	0.0	0.0	90	Clouds	overcast clouds	2012-10-02 11:00:00	4767
290.13	0.0	0.0	90	Clouds	overcast clouds	2012-10-02 12:00:00	5026
291.14	0.0	0.0	75	Clouds	broken clouds	2012-10-02 13:00:00	4918

```
df.shape
```

```
(48204, 9)
```

```
df['traffic_volume'].max()
```

```
7280
```

```
df['traffic_volume'].min()
```

```
0
```

Figura 20 - Descrição da base de dados

A seguir, iremos explorar as categorias que descrevem este conjunto de dados:

[10] Referências[3]

```
: df.columns

: Index(['holiday', 'temp', 'rain_1h', 'snow_1h', 'clouds_all',
  'weather_main',
         'weather_description', 'date_time', 'traffic_volume'],
        dtype='object')
```

Figura 21 - Categorias da base

Fica determinado então que a base apresenta as colunas ***'holiday', 'temp', 'rain_1h', 'snow_1h', 'clouds_all', 'weather_main', 'weather_description', 'date_time'***, que descrevem atributos relevantes para associação com a previsão do tráfego, e a classe ***'traffic_volume'*** como objeto de análise.

3.2.2 Dados discrepantes

Iniciamos então à analisar a discrepância entre os dados, conferindo se a base tem dados duplicados ou faltantes, e é constatado que 34 (trinta e quatro) registros estão duplicados, portanto, optamos por retirar os registros duplicados da base .

```
holiday                34
temp                   34
rain_1h                34
snow_1h                34
clouds_all             34
weather_main           34
weather_description    34
date_time              34
traffic_volume         34
dtype: int64
```

```
df.drop_duplicates(keep = 'first', inplace = True)
```

Figura 22 - Remoção de registros duplicados

```
pd.set_option('display.max_rows', None)
df.isnull().sum()
```

```
holiday                0
temp                   0
rain_1h                0
snow_1h                0
clouds_all             0
weather_main           0
weather_description    0
date_time              0
traffic_volume         0
dtype: int64
```

Figura 23 - Contagem de dados faltantes

A base não apresentou dados faltantes.

3.2.3 Explorando os *labels* de classificação

Na análise anterior, foi constatado que existem colunas com atributos nominais, à partir disso, fez-se necessário analisar quantos atributos diferentes cada uma dessas colunas apresentaria, abaixo uma descrição de todas as instancias encontradas em cada uma das colunas ***'holiday'*** que determina se o dia em questão seria ou não um feriado, ***'weather_main'*** que descreve uma categorização do clima, e a coluna ***'weather_description'*** que detalha melhor o estado do clima no dia em questão.

```
print(df['holiday'].unique())
print(df['weather_main'].unique())
print(df['weather_description'].unique())
```

```
['None' 'Columbus Day' 'Veterans Day' 'Thanksgiving Day' 'Christmas Day'
 'New Years Day' 'Washingtons Birthday' 'Memorial Day' 'Independence Day'
 'State Fair' 'Labor Day' 'Martin Luther King Jr Day']
['Clouds' 'Clear' 'Rain' 'Drizzle' 'Mist' 'Haze' 'Fog' 'Thunderstorm'
 'Snow' 'Squall' 'Smoke']
['scattered clouds' 'broken clouds' 'overcast clouds' 'sky is clear'
 'few clouds' 'light rain' 'light intensity drizzle' 'mist' 'haze' 'fog'
 'proximity shower rain' 'drizzle' 'moderate rain' 'heavy intensity rain'
 'proximity thunderstorm' 'thunderstorm with light rain'
 'proximity thunderstorm with rain' 'heavy snow' 'heavy intensity drizzle'
 'snow' 'thunderstorm with heavy rain' 'freezing rain' 'shower snow'
 'light rain and snow' 'light intensity shower rain' 'SQUALLS'
 'thunderstorm with rain' 'proximity thunderstorm with drizzle'
 'thunderstorm' 'Sky is Clear' 'very heavy rain'
 'thunderstorm with light drizzle' 'light snow'
 'thunderstorm with drizzle' 'smoke' 'shower drizzle' 'light shower snow'
 'sleet']
```

Figura 24 - Tipos de instancias nominais

Para melhor apreensão do algoritmo de regressão, é necessário então discretizar as instâncias, e transformar cada uma destas categorias nominais em dados numéricos, que variam de acordo com a quantidade de atributos distintos em cada coluna. Abaixo a exemplificação do uso da técnica de *Label Encoder*, que justamente visa atribuir um número a cada instância divergente dentro do *label*.

```
df['holiday'] = LabelEncoder().fit_transform(df['holiday'])
df['weather_main'] = LabelEncoder().fit_transform(df['weather_main'])
df['weather_description'] = LabelEncoder().fit_transform(df['weather_description'])
```

```
df.head()
```

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	date_time	traffic_volume
0	7	288.28	0.0	0.0	40	1	24	2012-10-02 09:00:00	5545
1	7	289.36	0.0	0.0	75	1	2	2012-10-02 10:00:00	4516
2	7	289.58	0.0	0.0	90	1	19	2012-10-02 11:00:00	4767
3	7	290.13	0.0	0.0	90	1	19	2012-10-02 12:00:00	5026
4	7	291.14	0.0	0.0	75	1	2	2012-10-02 13:00:00	4918

Figura 25 - Aplicação do Label Encoder

Na análise exploratória, nota-se a existência de um *label* que descreve a data e o horário em que foi medido o volume do tráfego. Foi necessário então realizar um tratamento subdividindo este *label* e separando em novas colunas com dia, mês, ano e hora, para melhor representação na aprendizagem regressora.
Abaixo é possível visualizar o resultado deste procedimento e as novas colunas que foram adicionadas a base, assim como a remoção da coluna que anteriormente se referia aos valores de data e horário.

```
data2['date_time'] = pd.to_datetime(data2['date_time'])
data2['year'] = data2['date_time'].dt.year
data2['month'] = data2['date_time'].dt.month
data2['day'] = data2['date_time'].dt.day
data2['hour'] = data2['date_time'].dt.hour
data2['weekday'] = data2['date_time'].dt.weekday


data2 = data2.drop(columns=['date_time'])

data2.head()
```

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	traffic_volume	date	hour	weekday	year	month	day
0	7	288.28	0.0	0.0	40	1	24	5545	2012-10-02	9	1	2012	10	2
1	7	289.36	0.0	0.0	75	1	2	4516	2012-10-02	10	1	2012	10	2
2	7	289.58	0.0	0.0	90	1	19	4767	2012-10-02	11	1	2012	10	2
3	7	290.13	0.0	0.0	90	1	19	5026	2012-10-02	12	1	2012	10	2
4	7	291.14	0.0	0.0	75	1	2	4918	2012-10-02	13	1	2012	10	2

Figura 26 - Separação da coluna Data/Hora

3.2.4 Normalização

Após a subdivisão da base de dados, decidimos por normalizar os valores para preparar a base para o treinamento.

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	traffic_volume	date	hour	weekday	year	month	day
0	0.636364	0.929726	0.0	0.0	0.40	1	24	5545	0.0	0.391304	0.166667	0.0	0.818182	0.033333
1	0.636364	0.933209	0.0	0.0	0.75	1	2	4516	0.0	0.434783	0.166667	0.0	0.818182	0.033333
2	0.636364	0.933918	0.0	0.0	0.90	1	19	4767	0.0	0.478261	0.166667	0.0	0.818182	0.033333
3	0.636364	0.935692	0.0	0.0	0.90	1	19	5026	0.0	0.521739	0.166667	0.0	0.818182	0.033333
4	0.636364	0.938949	0.0	0.0	0.75	1	2	4918	0.0	0.565217	0.166667	0.0	0.818182	0.033333

Figura 27 - Dados Normalizados

3.2.5 Correlação dos atributos

Ao verificar a correlação inicial considerando todos os atributos, percebe-se que o *label* que descreve a hora é o que maior apresenta correlação com o volume de tráfego, mesmo assim o valor ainda é baixo (0.35). Além disso, é possível constatar redundância em alguns atributos como a relação entre a descrição do tempo e a presença de nuvens no céu, ou os atributos de descrevem a data, o mês e o ano.

	holiday	temp	rain_1h	snow_1h	clouds_all	weather_main	weather_description	traffic_volume	date	hour	weekday	year	month	day
holiday	1.000000	-0.000473	0.000066	0.000432	0.007074	-0.004316	-0.002713	0.018677	-0.002872	0.026043	0.021454	-0.001426	-0.010110	0.007694
temp	-0.000473	1.000000	0.009070	-0.019756	-0.101968	-0.033434	-0.049437	0.130161	0.170399	0.112048	-0.007824	0.134916	0.223943	0.022677
rain_1h	0.000066	0.009070	1.000000	-0.000090	0.004818	0.009537	0.010777	0.004715	0.000613	0.003422	-0.006920	0.000443	0.001300	-0.002293
snow_1h	0.000432	-0.019756	-0.000090	1.000000	0.027934	0.036642	0.005104	0.000736	-0.000268	0.009852	-0.014929	-0.003514	0.020422	0.015798
clouds_all	0.007074	-0.101968	0.004818	0.027934	1.000000	0.500820	-0.341822	0.067138	-0.074492	0.054522	-0.039816	-0.072855	-0.009118	0.048425
weather_main	-0.004316	-0.033434	0.009537	0.036642	0.500820	1.000000	-0.127635	-0.040149	0.041203	-0.053599	-0.038731	0.036449	0.026351	0.023058
eather_description	-0.002713	-0.049437	0.010777	0.005104	-0.341822	-0.127635	1.000000	-0.067533	0.114941	-0.019611	0.031962	0.119352	-0.035962	-0.032775
traffic_volume	0.018677	0.130161	0.004715	0.000736	0.067138	-0.040149	-0.067533	1.000000	0.004249	0.352300	-0.149551	0.004697	-0.002480	-0.007760
date	-0.002872	0.170399	0.000613	-0.000268	-0.074492	0.041203	0.114941	0.004249	1.000000	-0.007499	-0.010818	0.988639	-0.009170	0.026308
hour	0.026043	0.112048	0.003422	0.009852	0.054522	-0.053599	-0.019611	0.352300	-0.007499	1.000000	-0.003808	-0.007561	0.001844	-0.009531
weekday	0.021454	-0.007824	-0.006920	-0.014929	-0.039816	-0.038731	0.031962	-0.149551	-0.010818	-0.003808	1.000000	-0.012396	0.010762	0.008627
year	-0.001426	0.134916	0.000443	-0.003514	-0.072855	0.036449	0.119352	0.004697	0.988639	-0.007561	-0.012396	1.000000	-0.158840	0.012091
month	-0.010110	0.223943	0.001300	0.020422	-0.009118	0.026351	-0.035962	-0.002480	-0.009170	0.001844	0.010762	-0.158840	1.000000	0.008686
day	0.007694	0.022677	-0.002293	0.015798	0.048425	0.023058	-0.032775	-0.007760	0.026308	-0.009531	0.008627	0.012091	0.008686	1.000000

Figura 28 - Matriz de Correlação

Decide-se então retirar da base os *labels* com redundância, ou correlação muito baixa com a classe de saída, que representa o tráfego.
Após a remoção dos *labels*, decidimos por conferir novamente a correlação entre os que restaram na base e a classe de saída, e o resultado final do processamento foi o ilustrado abaixo:

	holiday	temp	clouds_all	weather_main	weather_description	hour	weekday	traffic
0	0.636364	0.929726	0.40	0.1	0.648649	0.391304	0.166667	5545
1	0.636364	0.933209	0.75	0.1	0.054054	0.434783	0.166667	4516
2	0.636364	0.933918	0.90	0.1	0.513514	0.478261	0.166667	4767
3	0.636364	0.935692	0.90	0.1	0.513514	0.521739	0.166667	5026
4	0.636364	0.938949	0.75	0.1	0.054054	0.565217	0.166667	4918

```
]: correl = data3.corr()
   correl.style.background_gradient(cmap = 'coolwarm')
```

]:

	holiday	temp	clouds_all	weather_main	weather_description	hour	weekday	traffic
holiday	1.000000	-0.000473	0.007074	-0.004316	-0.002713	0.026043	0.021454	0.018677
temp	-0.000473	1.000000	-0.101968	-0.033434	-0.049437	0.112048	-0.007824	0.130161
clouds_all	0.007074	-0.101968	1.000000	0.500820	-0.341822	0.054522	-0.039816	0.067138
weather_main	-0.004316	-0.033434	0.500820	1.000000	-0.127635	-0.053599	-0.038731	-0.040149
weather_description	-0.002713	-0.049437	-0.341822	-0.127635	1.000000	-0.019611	0.031962	-0.067533
hour	0.026043	0.112048	0.054522	-0.053599	-0.019611	1.000000	-0.003808	0.352300
weekday	0.021454	-0.007824	-0.039816	-0.038731	0.031962	-0.003808	1.000000	-0.149551
traffic	0.018677	0.130161	0.067138	-0.040149	-0.067533	0.352300	-0.149551	1.000000

Figura 29 - Resultado da remoção de atributos

3.3 Treinamento e testes

3.3.1 Divisão da base

Partindo para a etapa de divisão da base e treinamento, a seguir ilustramos como a base de dados ficou inicialmente dividida separando o *label* referente ao tráfego dos demais.

```
x = data3.iloc[:, :7]
y = data3['traffic']
```

```
x.head()
```

	holiday	temp	clouds_all	weather_main	weather_description	hour	weekday
0	0.636364	0.929726	0.40	0.1	0.648649	0.391304	0.166667
1	0.636364	0.933209	0.75	0.1	0.054054	0.434783	0.166667
2	0.636364	0.933918	0.90	0.1	0.513514	0.478261	0.166667
3	0.636364	0.935692	0.90	0.1	0.513514	0.521739	0.166667
4	0.636364	0.938949	0.75	0.1	0.054054	0.565217	0.166667

```
y.head()
```

```
0    0.761676
1    0.620330
2    0.654808
3    0.690385
4    0.675549
Name: traffic, dtype: float64
```

Figura 30 - Divisão da base entre atributos e classe

Considerando o disposto na sessão do problema, testes com validação cruzada e diferentes algoritmos de regressão precisarão ser realizados, portanto, realizamos quatro procedimentos de forma randômica de divisão da base em treinamento e testes, para serem aplicados nos diferentes algoritmos, respeitando a proporção de 70% para treinamento e 30% para teste, diferente da questão anterior, pois esta base de dados é significantemente maior do que a do problema de classificação.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=9, shuffle = True)
x_train2, x_test2, y_train2, y_test2 = train_test_split(x, y, test_size=0.3, random_state=19, shuffle = True)

x_train3, x_test3, y_train3, y_test3 = train_test_split(x, y, test_size=0.3, random_state=25, shuffle = True)
x_train4, x_test4, y_train4, y_test4 = train_test_split(x, y, test_size=0.3, random_state=123, shuffle = True)
```

Figura 31 - Divisão da base em Treinamento e testes

3.3.2 Treinamento e Testes

O primeiro algoritmo a ser treinado foi a Árvore de Decisão [11] para regressão, executado duas vezes, com o primeiro e o segundo resultados do *test-split* ilustrado acima, seguido pelo kNN regressor, executado com o terceiro e quarto resultados do *test-split*, e os valores para k = 3 e k = 5, respectivamente:

```
classifier = tree.DecisionTreeRegressor(random_state = 42)
classifier.fit(x_train, y_train)
predictions = classifier.predict(x_test)


classifier2 = tree.DecisionTreeRegressor(random_state = 42)
classifier2.fit(x_train2, y_train2)
predictions2 = classifier2.predict(x_test2)
```

Figura 32 - Testes Iniciais Arvore de decisao

[11] Referências [11]

```
k1 = 3
k2 = 5
classifier3 = neighbors.KNeighborsRegressor(n_neighbors = k1)
classifier3.fit(x_train3, y_train3)
predictions3 = classifier.predict(x_test3)


classifier4 = neighbors.KNeighborsRegressor(n_neighbors = k2)
classifier4.fit(x_train4, y_train4)
predictions4 = classifier2.predict(x_test4)
```

Figura 33 - Testes Iniciais KNN

Além dos dois treinamentos realizados acima, foi implementado o algoritmo *SVM* para regressão, denominado *SVR*() na biblioteca "*sklearn"* para conferir como ele se comportaria utilizando as mesmas duas instâncias de entrada e testes do algoritmo de Árvore de Decisão.

```
classifier = svm.SVR(kernel='linear')
classifier.fit(x_train, y_train)
predictions = classifier.predict(x_test)


classifier2 = svm.SVR(kernel='linear')
classifier2.fit(x_train2, y_train2)
predictions2 = classifier2.predict(x_test2)
```

Figura 34 - Testes Iniciais SVM

3.3.3 Avaliação dos primeiros resultados

Para avaliação dos resultados, foram utilizadas as métricas de desempenho:

- Erro médio quadrático (MSE)
- Erro médio absoluto (MAD)
- Coeficiente de determinação (R2)

Para os dois primeiros (MSE) e (MAD), valores próximos de 0 (zero) indicam um modelo com menor erro, enquanto para o último um valor igual a 1 (um) indicam que o modelo se ajusta perfeitamente aos dados, o que poderia ser um indicativo de *overfitting*.

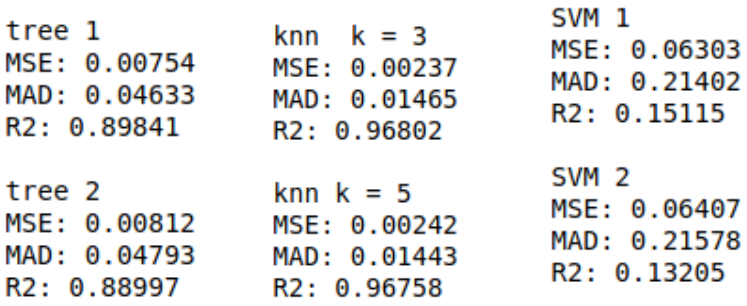


Figura 35 - Primeiros Resultados

Comparando as taxas de erros, o kNN apresentou a melhor performance para esse conjunto de dados, com erro muito próximo de zero, e a pior execução de aprendizado ficou com o SVM regressor.
Analisando a medida de R^2, o indicativo é de que não houve *overfitting* em nenhuma das predições, contudo o valor baixo do SVM para R^2 mostra que o algoritmo não se adaptou a base de dados da maneira que foi construída, e indica que futuramente outras abordagens podem ser consideradas como alternativas.

3.4 Validação cruzada

Para melhor compreender o desempenho dos algoritmos e comparar suas execuções ante os diferentes tipos de treinamento e teste, utilizaremos a validação cruzada aplicando o algoritimo *K-Means* [12] para dividir e executar diferentes vezes os procedimentos, podendo então calcular a média dos valores de suas execuções para representar com maior propriedade o desempenho dos regressores.
Para isso utilizaremos o valor de 10 (dez) divisões treinamentos e testes do regressor Árvore de Decisão e kNN.

Abaixo a representação da divisão da base e da validação cruzada:

[12] Referências [12]

	holiday	temp	clouds_all	weather_main	weather_description	hour	weekday	traffic
0	0.636364	0.929726	0.40	0.1	0.648649	0.391304	0.166667	5545.0
1	0.636364	0.933209	0.75	0.1	0.054054	0.434783	0.166667	4516.0
2	0.636364	0.933918	0.90	0.1	0.513514	0.478261	0.166667	4767.0
3	0.636364	0.935692	0.90	0.1	0.513514	0.521739	0.166667	5026.0
4	0.636364	0.938949	0.75	0.1	0.054054	0.565217	0.166667	4918.0

```
kf = model_selection.KFold(n_splits=10,random_state=42, shuffle=True)

knn = neighbors.KNeighborsRegressor(n_neighbors=3)
tee = tree.DecisionTreeRegressor()
```

```
splits = 10
results = []
fold = model_selection.KFold(n_splits = splits, random_state = 7, shuffle = True)
```

Figura 36 - Divisão para validação cruzada

Realizando novamente o treinamento e os testes, as 10 (dez) vezes determinadas anteriormente:

```
for train_index, test_index in kf.split(data3):
    x_train, x_test = x.loc[train_index], x.loc[test_index]
    y_train, y_test = y.loc[train_index], y.loc[test_index]
    knn.fit(x_train,y_train)
    resultknn = knn.predict(x_test)

    tee.fit(x_train,y_train)
    resulttree = tee.predict(x_test)
```

```
from sklearn.model_selection import cross_val_score

scores1 = cross_val_score(knn, x, y, cv = fold, scoring= 'neg_mean_squared_error')
scores2 = cross_val_score(tee, x, y, cv=fold, scoring='neg_mean_squared_error' )

scores3 = cross_val_score(knn, x, y, cv=fold, scoring='neg_mean_absolute_error' )
scores4 = cross_val_score(tee, x, y, cv=fold, scoring='neg_mean_absolute_error')


scores5 = cross_val_score(knn, x, y, cv=fold, scoring='r2')
scores6 = cross_val_score(tee, x, y, cv=fold,scoring='r2')
```

Figura 37 - Treinamento sob Validação cruzada

Podemos finalmente calcular a média dos resultados das execuções e comparar à execução anterior. Nesta execução, a função *cross_validation_score* inverte os valores de MSE e MAE que são representados com um número negativo, neste caso, os valores próximos de 0 são indicativos de melhores resultados.
Abaixo, a representação da média das métricas de avaliação de todas as execuções realizadas com cross validation.

```
mean sq error
 knn: -241012.50481420424
tree:  -78390.97107463916

 mean absolut eror
 knn:  -229.0480470196998
tree:  -136.00446342961834

R²

 knn: 0.9389531698832323
tree: 0.9811480229780569
```

Figura 38 - Media dos resultados Validação cruzada

É notável que a validação cruzada impactou diretamente as métricas de erro, e não impactou com significância a medida de "R²" das execuções anteriores, que ainda indica não haver *overfitting* na adaptação dos modelos.

4. Terceiro problema - Clusterização

4.1 Metodologia

O terceiro problema dessa avaliação visa aplicar algoritmos de clusterização a uma base de dados que contém atributos referentes ao comportamento em uma rede social, mais especificamente, serão executados os algoritmos de agrupamento *K-means* [13] e *Hierárquico* [14] com diferentes valores de clusters. O objetivo é encontrar a melhor solução na formação de *clusters* pelos algoritmos.

4.1.1 Pré-processamento dos dados

Semelhante aos problemas anteriores, as etapas a seguir visam minimizar a discrepância e desbalanceamento, caso exista, na base de dados, a fim de melhorar a precisão da clusterização.
Na primeira analise dos dados nos deparamos com 4 (quatro) colunas de atributos não especificados, e completamente preenchidas com valores nulos, como pode ser observado na imagem abaixo:

[13] Referências[12]
[14] Referências [13]

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432	92	3	
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150	0	0	
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204	21	1	
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	111	0	0	111	0	0	
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	213	0	0	204	9	0	

```
df.info()
```

```
 0   status_id          7050 non-null   object
 1   status_type        7050 non-null   object
 2   status_published   7050 non-null   object
 3   num_reactions      7050 non-null   int64
 4   num_comments       7050 non-null   int64
 5   num_shares         7050 non-null   int64
 6   num_likes          7050 non-null   int64
 7   num_loves          7050 non-null   int64
 8   num_wows           7050 non-null   int64
 9   num_hahas          7050 non-null   int64
 10  num_sads           7050 non-null   int64
 11  num_angrys         7050 non-null   int64
 12  Column1            0 non-null      float64
 13  Column2            0 non-null      float64
 14  Column3            0 non-null      float64
 15  Column4            0 non-null      float64
dtypes: float64(4), int64(9), object(3)
```

Figura 39 - Tipos de dados na base

As últimas quatro colunas não apresentam nenhum valor diferente de nulo, para as demais, existem 7050 (sete mil e cinquenta) atributos descritos.

```
print(len(df['status_id'].unique()))
print(len(df['status_published'].unique()))
print(len(df['status_type'].unique()))
print(len(df['Column1'].unique()))
print(len(df['Column2'].unique()))
print(len(df['Column3'].unique()))
print(len(df['Column4'].unique()))
print(len(df['status_type'].unique()))
```

```
6997
6913
4
1
1
1
1
4
```

Figura 40 - Contagem de dados diferentes por categoria

4.1.2 Dados discrepantes

Ao explorar a variação individual de tipos de atributos por coluna, é notável que as duas primeiras colunas ***"status_id"*** e ***"status_published"*** possuem quase sete mil instancias diferentes, e que existem 4 (quatro) tipos de tipos de *status* descritos na coluna ***"status_type",*** assim como a confirmação de que as 4 colunas finais da base estão completamente preenchidas com valores nulos repetitivos.

Decidimos então remover as colunas com altos valores de atributos divergentes, ou completamente irrelevantes e nulos.

```
In [305]: df.drop(['Column1', 'Column2', 'Column3', 'Column4', 'status_id', 'status_published'], axis=1, inplace=True)

In [306]: df.head()

Out[306]:
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	video	529	512	262	432	92	3	1	1	0
1	photo	150	0	0	150	0	0	0	0	0
2	video	227	236	57	204	21	1	1	0	0
3	photo	111	0	0	111	0	0	0	0	0
4	photo	213	0	0	204	9	0	0	0	0

```
In [307]: df.shape
Out[307]: (7050, 10)
```

Figura 41 - Remoção de atributos

O procedimento de remoção resulta em 10 (dez) *labels* finais. O próximo passo é identificar e remover os valores duplicados, como foi feito nas abordagens anteriores.

```
pd.set_option('display.max_rows', None)
df[df.duplicated(keep=False)].shape
```

```
(2521, 10)
```

```
df.drop_duplicates(keep = 'first', inplace = True)
```

```
df.shape
```

```
(4987, 10)
```

Figura 42 - Removendo duplicatas

A base original continha muitos valores duplicados, a remoção desses atributos resultou numa diminuição das instâncias para 4987 (quatro mil novecentos e oitenta e sete) atributos.

4.1.3 Explorando os *labels*

Da mesma maneira que a abordagem nos problemas anteriores, fez-se necessário aplicar o processamento de *Label Encoder*, para discretizar as categorias com atributos nominais para numéricos, procedimento ilustrado a seguir :

```
le = LabelEncoder()

x['status_type'] = le.fit_transform(x['status_type'])

y = le.transform(y)
```

```
x.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	3	529	512	262	432	92	3	1	1	0
1	1	150	0	0	150	0	0	0	0	0
2	3	227	236	57	204	21	1	1	0	0
3	1	111	0	0	111	0	0	0	0	0
4	1	213	0	0	204	9	0	0	0	0

Figura 43 - Discretização de atributos nominais

4.1.4 Correlação

Com todos os atributos da base descritos numericamente, podemos então verificar a correlação entre eles e a classe de saída "***status type***":

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
status_type	1.000000	0.083964	0.346349	0.424454	0.042570	0.421731	0.096132	0.186597	0.088522	0.146692
num_reactions	0.083964	1.000000	0.115832	0.216485	0.994276	0.268827	0.247059	0.154776	0.055539	0.111795
num_comments	0.346349	0.115832	1.000000	0.629457	0.064419	0.505862	0.151545	0.313187	0.245972	0.227398
num_shares	0.424454	0.216485	0.629457	1.000000	0.133215	0.813046	0.401438	0.388001	0.204206	0.320390
num_likes	0.042570	0.994276	0.064419	0.133215	1.000000	0.166203	0.182684	0.096318	0.030460	0.070990
num_loves	0.421731	0.268827	0.505862	0.813046	0.166203	1.000000	0.504224	0.498701	0.211859	0.384172
num_wows	0.096132	0.247059	0.151545	0.401438	0.182684	0.504224	1.000000	0.281244	0.083116	0.184150
num_hahas	0.186597	0.154776	0.313187	0.388001	0.096318	0.498701	0.281244	1.000000	0.144524	0.216208
num_sads	0.088522	0.055539	0.245972	0.204206	0.030460	0.211859	0.083116	0.144524	1.000000	0.134941
num_angrys	0.146692	0.111795	0.227398	0.320390	0.070990	0.384172	0.184150	0.216208	0.134941	1.000000

Essa etapa foi fundamental para a abordagem deste problema, pois a matriz de correlação deixa evidente a redundância de alguns atributos com correlação superior a 99%, o que indica que os *labels* descrevem praticamente o mesmo tipo de dado.

A partir da avaliação dessas categorias, foi possível diminuir ainda mais as colunas de atributos descritores da base, para tanto, utilizou-se o critério de excluir uma das categorias que estejam altamente relacionadas entre si, e manter aquela que apresentar maior correlação com a classe ***"status_type".*** Abaixo está ilustrado este procedimento, e a base resultante para as próximas etapas.

```
x.drop(['num_likes'], axis=1, inplace=True)
```

```
x.drop(['num_loves'], axis=1, inplace=True)
```

```
x.head()
```

	status_type	num_reactions	num_comments	num_shares	num_wows	num_hahas	num_sads	num_angrys
0	3	529	512	262	3	1	1	0
1	1	150	0	0	0	0	0	0
2	3	227	236	57	1	1	0	0
3	1	111	0	0	0	0	0	0
4	1	213	0	0	0	0	0	0

Figura 44 - Base final apos a remoção de atributos altamente correlacionados

4.2 Clusterização

4.2.1 Clusterização dos dados originais

A seguir iniciaremos a etapa de aplicação dos algoritmos de clusterização. Para tanto a base novamente foi dividida em atributos e classe, "*x*" e "*y*" respectivamente.
Para análise, foram aplicados o algoritmo de K-*means*, o Hierárquico Simples e o Hierárquico Completo, e sua definição esta ilustrada na imagem a seguir:

```
intera = [i**2 for i in range (2, 10)]

for n in intera:

    kmeans_r = cluster.KMeans(n_clusters=n,init ="random").fit(x)
    print('inertia para', n , 'clusters: ' , kmeans_r.inertia_)

    labelsk = kmeans_r.labels_
    correct_labelsk = sum(y == labelsk)
    print("Result: %d out of %d samples were correctly labeled." % (correct_labelsk, y.size))

    ######################



    cluster_kmeans = x.assign(cluster_numero = kmeans_r.predict(x))


    hierarquico_simples = cluster.AgglomerativeClustering(n_clusters=n, linkage="single")
    hierarquico_completo = cluster.AgglomerativeClustering(n_clusters=n, linkage="complete")

    cluster_hierarquico_simples = x.assign(cluster_numero = hierarquico_simples.fit_predict(x))
    cluster_hierarquico_completo = x.assign(cluster_numero = hierarquico_completo.fit_predict(x))


    label_kmeans.append(cluster_kmeans)
    labels_hiesimples.append(cluster_hierarquico_simples)
    labels_hiecompleto.append(cluster_hierarquico_completo)

    silhueta.append({ "Número de Clusters": n,
                      "K-Means - aleatório": metrics.silhouette_score(cluster_kmeans,labels = cluster_kmeans['clust
                     "Hierarquico Simples": metrics.silhouette_score(cluster_hierarquico_simples,labels = cluster_h
                     "Hierarquico Completo": metrics.silhouette_score(cluster_hierarquico_completo,labels = cluster
```

Figura 45 - Algoritmos de Clusterização

A execução buscou variar o número de clusters entre 4 (quatro) e 81 (oitenta e um) clusters, de maneira não-linear.
O resultado do processamento dos dados originais não foi satisfatório, as medidas de *Inertia* e *Silhouette score*, foram os parâmetros utilizados para análise, e como é possível observar na figura abaixo, poucos atributos foram corretamente agrupados.

```
inertia para 4 clusters:  1573900246.8312376
Result: 139 out of 4987 samples were correctly labeled.
inertia para 9 clusters:  480079761.4501137
Result: 78 out of 4987 samples were correctly labeled.
inertia para 16 clusters:  217625735.65231943
Result: 157 out of 4987 samples were correctly labeled.
inertia para 25 clusters:  143941515.85071996
Result: 68 out of 4987 samples were correctly labeled.
inertia para 36 clusters:  108382069.35438229
Result: 44 out of 4987 samples were correctly labeled.
inertia para 49 clusters:  99601818.34493768
Result: 5 out of 4987 samples were correctly labeled.
inertia para 64 clusters:  97224889.86195348
Result: 151 out of 4987 samples were correctly labeled.
inertia para 81 clusters:  91084769.51451546
Result: 31 out of 4987 samples were correctly labeled.
```

Figura 46 - Resultado dos primeiros agrupamentos *Inertia*

Sucintamente, o valor da *inertia* descreve quão longe os pontos estão dentro de um cluster, portanto, o desejado para esta medida é um valor pequeno, mais próximo de zero do que o que foi mensurado acima.

A *Silhouette score* determina as distância entre os pontos de um *cluster* a outro, e pode variar de -1 (um negativo) á 1 (um positivo). Valores próximos à 1 (um) indicam que a amostra está longe dos aglomerados vizinhos. Um valor 0 (zero) indica que a amostra está dentro ou muito perto do limite de decisão entre dois clusters vizinhos e valores negativos indicam que essas amostras podem ter sido atribuídas ao cluster errado.
Abaixo um gráfico para representar as medidas de *Silhouette:*

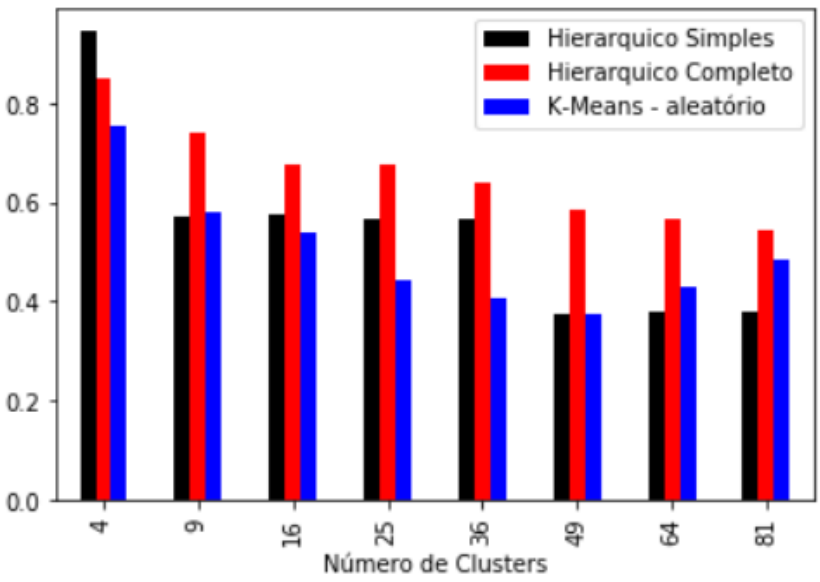


Figura 47 - *Silhouette score* para os primeiros agrupamentos

Percebe-se então que o aumento do número de clusters tem impacto negativo no desempenho dos algoritmos, e que para este conjunto de dados não normalizado o Hierárquico simples apresentou os melhores resultados.

4.2.2 Clusterização dos dados Normalizados

Como sugerido pelo problema, vamos avaliar a seguir os impactos da normalização dos dados no resultado do agrupamento.

```
normal = preprocessing.RobustScaler()

normalizacao = pd.DataFrame(normal.fit_transform(x),columns = x.columns.values)

normalizacao.head()
```

	status_type	num_reactions	num_comments	num_shares	num_wows	num_hahas	num_sads	num_angrys
0	1.0	1.605691	5.770115	16.3125	3.0	1.0	1.0	0.0
1	0.0	0.065041	-0.114943	-0.0625	0.0	0.0	0.0	0.0
2	1.0	0.378049	2.597701	3.5000	1.0	1.0	0.0	0.0
3	0.0	-0.093496	-0.114943	-0.0625	0.0	0.0	0.0	0.0
4	0.0	0.321138	-0.114943	-0.0625	0.0	0.0	0.0	0.0

Figura 48 - Dados normalizados

Após a normalização dos dados, vamos executar novamente os algoritmos e medir seu desempenho sob a base normalizada:

```
inertia para 4 clusters:  589999.6724437145
Result: 2803 out of 4987 samples were correctly labeled.
inertia para 9 clusters:  335299.82901501155
Result: 587 out of 4987 samples were correctly labeled.
inertia para 16 clusters:  224782.11682838926
Result: 31 out of 4987 samples were correctly labeled.
inertia para 25 clusters:  168576.95907887715
Result: 2028 out of 4987 samples were correctly labeled.
inertia para 36 clusters:  133597.8474634226
Result: 15 out of 4987 samples were correctly labeled.
inertia para 49 clusters:  118232.87636223473
Result: 19 out of 4987 samples were correctly labeled.
inertia para 64 clusters:  127952.89229290557
Result: 65 out of 4987 samples were correctly labeled.
inertia para 81 clusters:  93623.87447130792
Result: 89 out of 4987 samples were correctly labeled.
```

Figura 49 - Medidas de desempenho para dados normalizados

Para esta execução os valores de *inertia* continuam altos, e conforme o numero de clusters aumenta o algoritmo praticamente não consegue agrupa-los.
Para 4 (quatro) clusters, temos um total de 2803 (vinte e oito mil e três) clusters de 4987 (quatro mil novecentos e oitenta e sete) corretamente agrupados, que é o nosso melhor resultado até o momento.

Analisando a *Sillhuet score* à seguir:

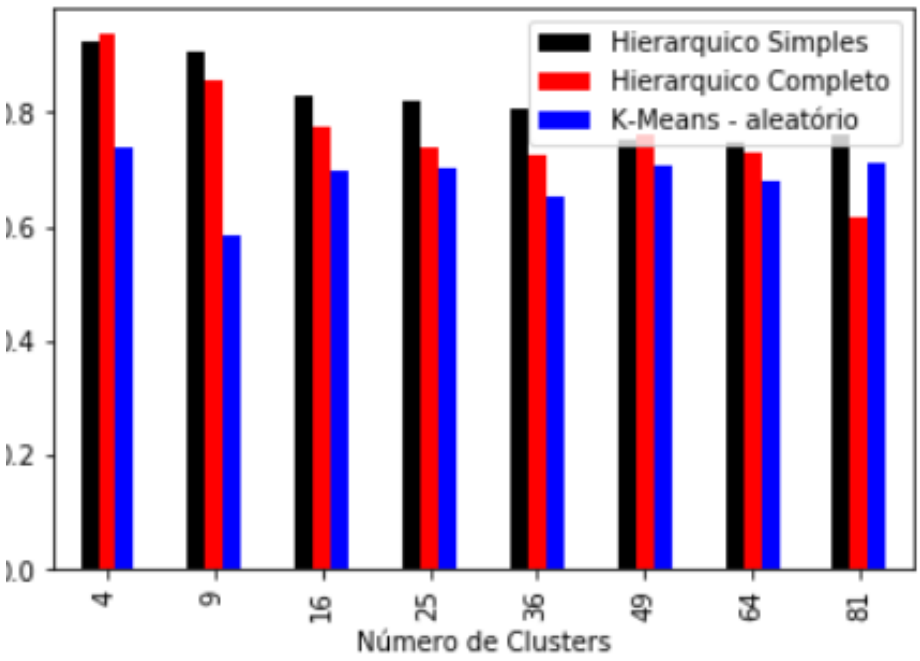


Figura 50 - Silhouette Score para dados normalizados

Percebe-se que a normalização dos dados impactou positivamente o desempenho de todos os algoritmos, e que na primeira execução o Hierárquico completo conseguiu superar o Simples, disputando o primeiro lugar com este nas próximas execuções e vencendo novamente para 49 clusters.
O k-means, teve uma melhora significativa e pareceu se estabilizar em relação a performance a partir dos 16 clusters.

Para visualização dos clusters foram aplicados PCA e TSNE , e as ilustrações podem ser visualizadas abaixo :

```
In [343]: sns.scatterplot(x = pcametrics[0], y = pcametrics[1], data = pcametrics)
Out[343]: <matplotlib.axes._subplots.AxesSubplot at 0x7f930a96aa50>
```

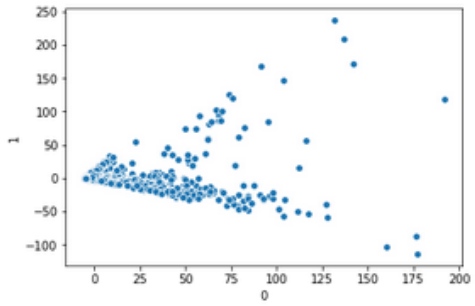


Figura 51 - Visualização dos clusters com PCA

```
In [345]: sns.scatterplot(x = pcametrics[0], y = pcametrics[1], hue = labelnor_kmeans[0][
Out[345]: <matplotlib.axes._subplots.AxesSubplot at 0x7f930a8defd0>
```

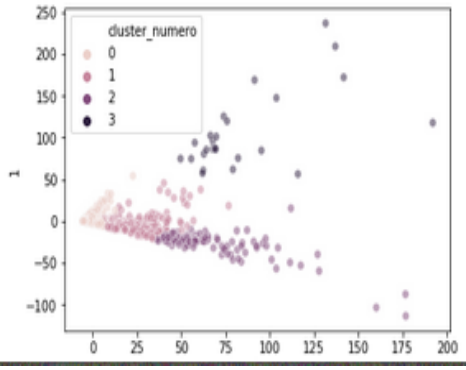


Figura 52 – PCA Clusters K-Means 1

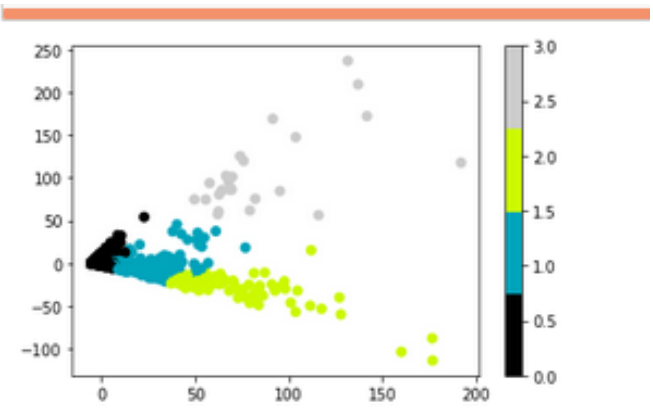


Figura 53 - PCA Clusters K-Means 2

```
: sns.scatterplot(x = pcametrics[0], y = pcametrics[1],hue = labelsnor_hiesimples[0][

: <matplotlib.axes._subplots.AxesSubplot at 0x7f930ab58910>
```

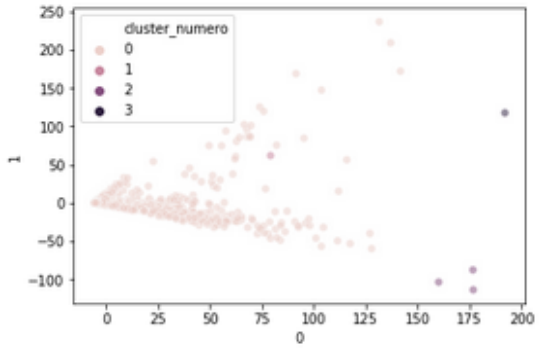


Figura 54 – PCA Clusters Hier. Simples 1

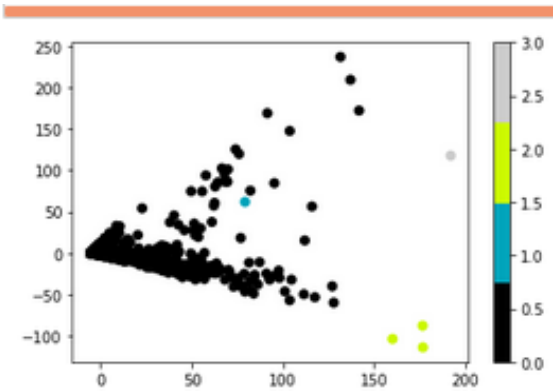
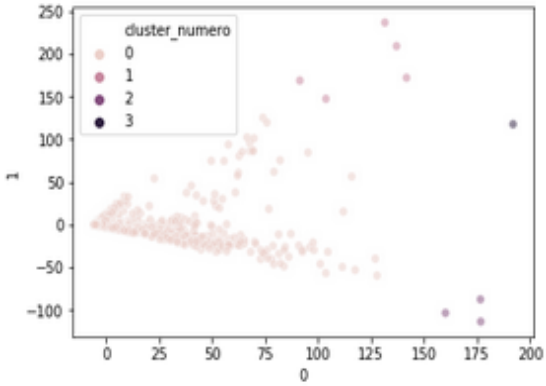


Figura 55 - PCA Clusters Hier. Simples 2

```
347]: sns.scatterplot(x = pcametrics[0], y = pcametrics[1],hue = labelsnor_hiecompleto[0]
```

```
347]: <matplotlib.axes._subplots.AxesSubplot at 0x7f930a7fc310>
```




```
350]: plt.figure()
plt.scatter(pcametrics[0], pcametrics[1], c=labelsnor_hiecompleto[0]['cluster_numero
plt.colorbar()
plt.show()
```

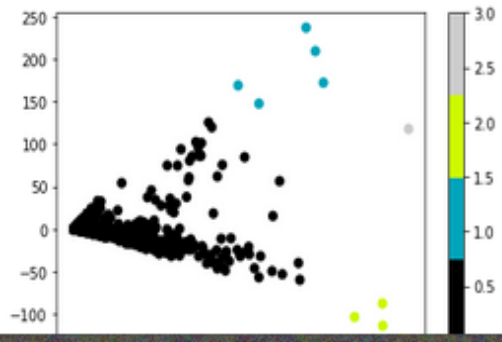


Figura 56 - PCA Clusters Hier. Completo 1 e 2

```
In [344]: tsne_df.head()
```

Out[344]:

	0	1
0	29.562815	-45.927055
1	28.748323	52.370358
2	2.020931	-17.525232
3	38.378956	46.660995
4	13.034575	61.337158

```
In [292]: sns.scatterplot(x = tsne_df[0], y = tsne_df[1], data = tsne_df, legend = 'brief')
Out[292]: <matplotlib.axes._subplots.AxesSubplot at 0x7f93103be510>
```

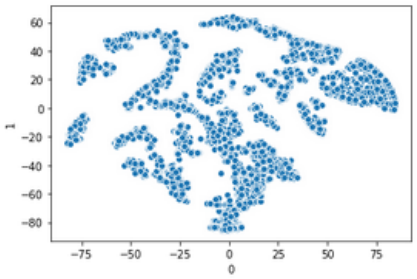


Figura 57- TSNE Clusters

```
sns.scatterplot(x = tsne_df[0], y = tsne_df[1], hue = labelnor_kmeans[0]['
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f931063bd10>
```

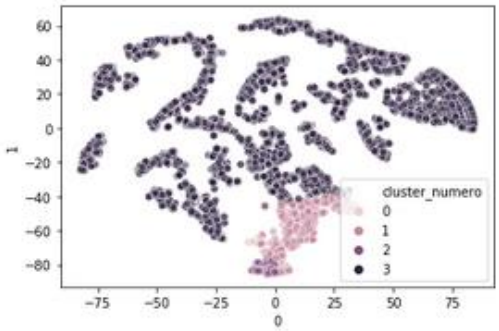


Figura 58 - TSNE k-means

```
sns.scatterplot(x = tsne_df[0], y = tsne_df[1],hue = labelsnor_hiesimples[0]['c
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f93101d5210>
```

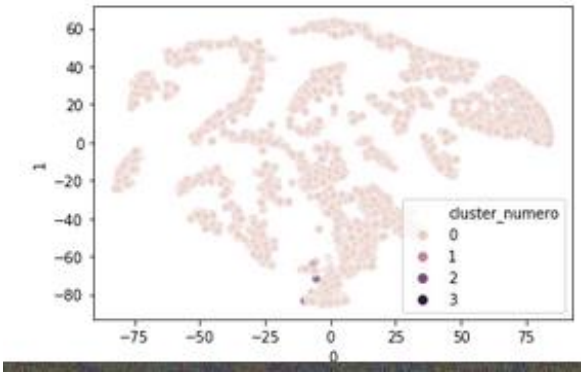


Figura 59-TSNE Hier. Simples

```
: sns.scatterplot(x = tsne_df[0], y = tsne_df[1],hue = labelsnor_hiecompleto[0][
```

```
: <matplotlib.axes._subplots.AxesSubplot at 0x7f93103ac9d0>
```

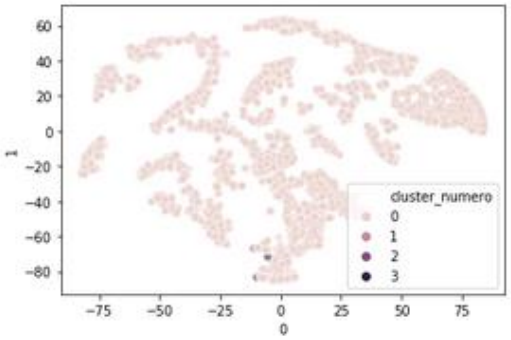


Figura 60- TSNE Hier. Completo

5. Conclusão

O presente trabalho se mostrou uma ferramenta poderosa de apreensão do conteúdo da disciplina de inteligência artificial, de modo que ao realiza-lo o pesquisador teve oportunidade de implementar utilizando problemas reais, as técnicas aprendidas em sala de aula.

É reconhecida a riqueza do aprendizado a partir do desenvolvimento deste trabalho, e o fascínio pela área de ciência de dados é inevitável. Em todos os módulos aqui discutidos existiram problemas de construção das bases de dados e de execuções dos algoritmos para cada caso, que foram devidamente expostos e explorados ao longo do desenvolvimento. Dessa forma, concluímos que os problemas reais são significativamente únicos, e cada abordagem deve distinguir-se das outras para conseguir melhores resultados, paralelamente a isto, compreende-se que podem existir melhores formas de abordar os problemas aqui descritos, e futuramente explora-los agregará ainda mais o conhecimento aqui fundido.

Referências

[1] WIKIPEDIA. **Supervised learning**, 2020. Disponível em: <https://en.wikipedia.org/wiki/Supervised_learning >. Acesso em: 03 de mar. de 2020.

[2] WIKIPEDIA. **Multi-label classification**, 2020. Disponível em: <https://en.wikipedia.org/wiki/Multi-label_classification>. Acesso em: 02 de mar. de 2020.

[3] TOWARDS DATA SCIENCE. **Supervised Learning: Basics of Linear Regression**, 2019. Disponível em: <https://towardsdatascience.com/supervised-learning-basics-of-linear-regression-1cbab48d0eba>. Acesso em: 02 de mar. de 2020.

[4] WIKIPEDIA. **Clustering**, 2018. Disponível em: <https://pt.wikipedia.org/wiki/Clustering>. Acesso em: 02 de mar. de 2020.

[5] WIKIPEDIA. **Multiclass classification**, 2020. Disponível em: <https://en.wikipedia.org/wiki/Multiclass_classification>. Acesso em: 02 de mar. de 2020.

[6] BISHOP, CHRISTOPHER M. (2006). **Pattern Recognition and Machine Learning**. Springer.

[7] WIKIPEDIA. **Regressão logística**, 2019. Disponível em: <https://pt.wikipedia.org/wiki/Regress%C3%A3o_log%C3%ADstica>. Acesso em: 03 de mar. de 2020.

[8] TOWARDS DATAS CIENCE. **Deep dive into multi-label classification**, 2018. Disponível em: <https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff>. Acesso em: 03 de mar. de 2020.

[9] TOWARDS DATAS CIENCE. **Support Vector Machine — Introduction to Machine Learning Algorithms**, 2018. Disponível em: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>. Acesso em: 03 de mar. de 2020.

[10] WIKIPEDIA. **k-nearest neighbors algorithm,** 2020. Disponível em: <https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm >. Acesso em: 03 de mar. de 2020.

[11] LUCIDCHART. **O que é um diagrama de árvore de decisão?** . Disponível em: <https://www.lucidchart.com/pages/pt/o-que-e-arvore-de-decisao#section_5>. Acesso em: 04 de mar. de 2020.

[12] MINERANDODADOS. **Entenda o Algoritmo K-means**. 2018. Disponível em: <https://minerandodados.com.br/entenda-o-algoritmo-k-means/>. Acesso em: 04 de mar. de 2020.

[13] OPERDATA. **Análise de Cluster**. 2019. Disponível em: <https://operdata.com.br/blog/analise-de-cluster/>. Acesso em: 04 de mar. de 2020.

[14] TOWARDS DATAS CIENCE. **Train/Test Split and Cross Validation in Python**, 2019. Disponível em: <https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6>. Acesso em: 03 de mar. de 2020.

[15] DATACAMP. **K-Means Clustering in Python with scikit-learn**, 2018. Disponível em: <https://www.datacamp.com/community/tutorials/k-means-clustering-python >. Acesso em: 04 de mar. de 2020.