Adriana Zaniol
Kalel Chaves
Rafael Smiderle

Resumo

Neste artigo iremos apresentar todo o processo pré-processamento, análise, e aplicação de algoritmos de aprendizado de máquina não supervisionados de agrupamento sobre um conjunto de dados, dataset, extraídos da plataforma Kaggle. O dataset apresenta informações sobre usuários de cartão de crédito, no dataset temos informações pessoais, como número de filhos, idade, tempo em que está empregado, status sobre pagamentos entre outros. Além de uma análise exploratória, foram realizados diversos tratamentos, sobre dados faltantes e o grande desafio foi lidar com o desbalanceamento das informações, visto que as informações sobre bons pagadores supera e muito os maus pagadores, tendo em vista que a ideia era predizer bons pagadores, facilmente essa dificuldade foi superada.
Foram utilizados diversos algoritmos de agrupamento como o K-Means em conjunto com algoritmos para validação no caso do K-means SSE porém outras métricas foram utilizadas também como Davies Bouldin e Calinski Harabasz.

Introdução

A ciência de dados é um conjunto de fundamentos e técnicas que tem como objetivo nos revelar o conhecimento sobre as informações que possuímos em um determinado conjunto de dados. O processo consiste em coletar, preparar e processar informações com a finalidade de obter conhecimento. Neste artigo vamos analisar um dataset de usuários de cartão de crédito, nele poderemos ver bons e maus pagadores.
Contextualizando, os métodos de pontuação de crédito são um método de controle de risco comum no setor financeiro. Ele usa informações pessoais e dados enviados por solicitantes de cartão de crédito para prever a probabilidade de inadimplências futuras e empréstimos. O banco pode decidir se emite um cartão de crédito ao requerente bem como gerir o seu limite. As classificações sobre o crédito podem quantificar objetivamente a magnitude do risco. De um modo geral, a pontuação de crédito é baseada em dados históricos.

Uma vez encontrando grandes flutuações econômicas, os modelos mais antigos podem perder seu poder preditivo original e com a constante evolução no poder de processamento dos computadores, um método mais automatizado passa a ser mais comum para classificação de crédito.

Atualmente, com o desenvolvimento de algoritmos de aprendizado de máquina. Podemos usufruir destes algoritmos para realizar a predição de bons e maus pagadores. É o que vamos poder conferir neste artigo.

Materiais e Métodos

A coleta de dados foi realizada através da plataforma Kaggle. O dataset utilizado foi o https://www.kaggle.com/rikdifos/credit-card-approval-prediction e representa a base de clientes e o histórico de faturas de uma operadora de cartões de crédito.
Após a coleta, os dois datasets, *"application_record.csv"* e *"credit_record.csv"*, foram carregados para um ambiente de notebook, Jupyter Lab ou Google Colab utilizando a linguagem Python.
Após a carga, realizamos a junção dos dois datasets pela coluna ID conforme descrito na documentação do dataset. Após, foi iniciada uma análise buscando por ruídos nos dados, como informações não preenchidas, outliers e dados desbalanceados.
Em uma primeira análise já pudemos constatar o desbalanceamento do dataset. A quantidade de bons pagadores é muito superior à quantidade de maus pagadores. Para diferenciar um bom pagador de um mau pagador foi considerado o status de pagamento do da sua última fatura, sendo os status descritos na tabela abaixo.

STATUS_0	1-29 dias em atraso
STATUS_1	30-59 dias em atraso
STATUS_2	60-89 dias em atraso
STATUS_3	90-119 dias em atraso
STATUS_4	120-149 dias em atraso
STATUS_5	Dívidas vencidas ou incobráveis, baixas por mais de 150 dias
STATUS_C	Quitado no mês de vencimento
STATUS_X	Nenhum uso para o mês

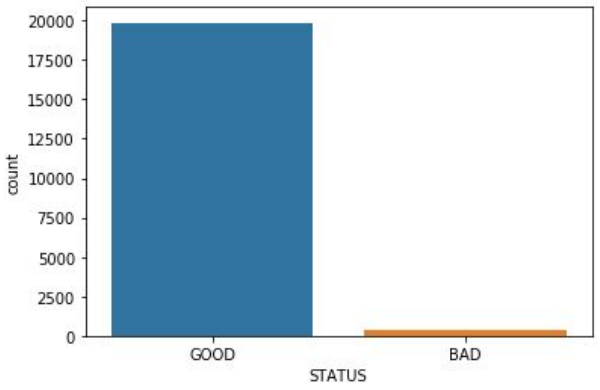
Pré-processamento

Inicialmente realizamos o filtro do dataset, pois decidimos centrar a análise nos dados do último mês. Portanto, filtramos as 777.715 instâncias do dataset original mantendo apenas aquelas cuja feature MONTHS_BALANCE fosse igual a "0" que, conforme a documentação do dataset, indica que os dados são do mês corrente. Dessa forma centramos nossa análise nas 24.672 instâncias mais recentes.
Prosseguimos com o tratamento de dados faltantes e remoção de features irrelevantes. As informações da coluna "*OCCUPATION_TYPE*" haviam dados faltantes, os mesmos foram preenchidos com "*uninformed*". Haviam também estados civis parecidos, irrelevantes para ficarem separados, para isso foi feito um *replace* na coluna *NAME_FAMILY_STATUS* e todas as classes que possuíam a descrição "*Civil marriage*" foram alteradas para "*Married*". Também foram removidas features com variância zero por entender que seriam irrelevantes para a classificação além das colunas ID e MONTHS_BALANCE que são irrelevantes para a análise e foram úteis apenas na junção e filtro do dataset.

Para fins de agrupamento, classificando categoricamente como bons e maus pagadores, realizamos o seguinte agrupamento e criação de novos status. Os status “C” e “0” foram considerados bons pagadores, para os quais adotamos o status “GOOD”. Já os status “1”, “2”, “3”, “4” e “5” foram considerados maus pagadores, para os quais atribuímos o status “BAD”. Os registros com status “X” foram desconsiderados pois são clientes que não fizeram uso do cartão de crédito no período analisado. Após isso geramos um histograma e visualmente percebemos o desbalanceamento.

```
sns.countplot(df_merge['STATUS'],
              order=df_merge['STATUS'].value_counts().index)
```

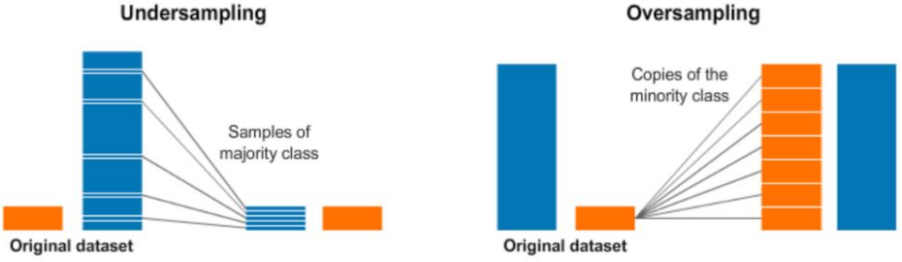
```
<matplotlib.axes._subplots.AxesSubplot at 0x26d1eb5b748>
```



Após estes tratamentos, realizamos a transformações de todas as colunas em categóricas utilizando o método “*get_dummies()”* da biblioteca *pandas*. Com isso, foram geradas algumas features excludentes entre si, como por exemplo “CODE_GENDER_F” e “CODE_GENDER_M”, “STATUS_GOOD” e “STATUS_BAD”. Em ambos os casos, mantivemos apenas uma das variáveis.

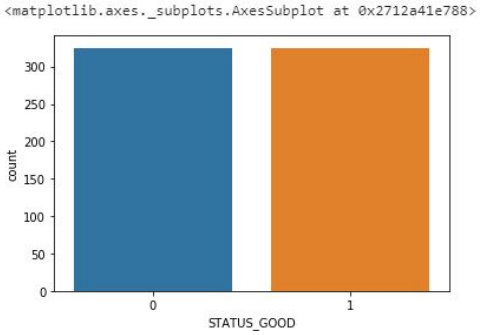
```
#eliminando features que são inversas entre si
dummies.drop(columns=['CODE_GENDER_M'], inplace=True) #foi mantida a CODE_GENDER_F
dummies.drop(columns=['STATUS_BAD'], inplace=True) #foi mantida a STATUS_GOOD
dummies.drop(columns=['FLAG_OWN_CAR_N','FLAG_OWN_REALTY_N'], inplace=True)
dummies.rename(columns={'FLAG_OWN_CAR_N' : 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY_N': 'FLAG_OWN_REALTY'}, inplace=True)
```

Para resolver o problema do desbalanceamento de dados, avaliamos algumas técnicas, *undersampling* e de *oversampling*. Técnicas de *oversampling* são aquelas técnicas que irão balancear os dados conforme a classe minoritária, no caso teremos uma perda de dados com relação ao status “GOOD”. Já as técnicas de *oversampling*, são técnicas que tratam o desbalanceamento através da classe majoritária, neste caso são gerados dados baseados nos existentes para termos uma quantidade balanceada de informações de bons e maus pagadores. A imagem abaixo exemplifica os dois métodos.



Dentro de cada técnica para tratar o desbalanceamento, temos alguns métodos na biblioteca "*imblearnb"* que podem nos ajudar. Para realizar o *undersample* utilizamos NearMiss. O mesmo foi realizado com métodos de *oversample* como SMOTE e ADASYN. Os resultados foram avaliados através de um modelo de regressão logística que foi aplicado no dataset inicial e também em cada um dos datasets resultantes das técnicas de balanceamento aplicadas par então comparar a eficácia de cada um dos modelos resultantes na predição da variável STATUS e assim selecionar a melhor técnica para esse dataset.

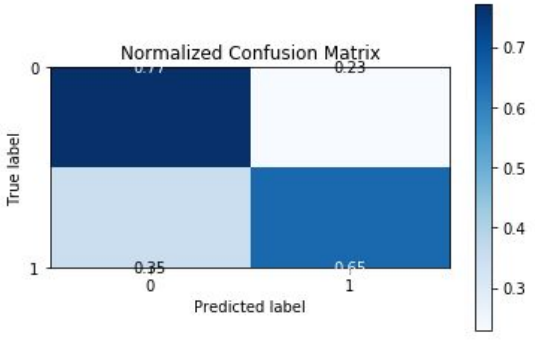
Dataset resultante após o *undersample*:



Avaliação da performance do dataset para a predição de bons pagadores após o *undersample*:

```
df_undersampled = X1.merge(y1, left_index=True, right_index=True)
testa_modelo(df_undersampled)
```

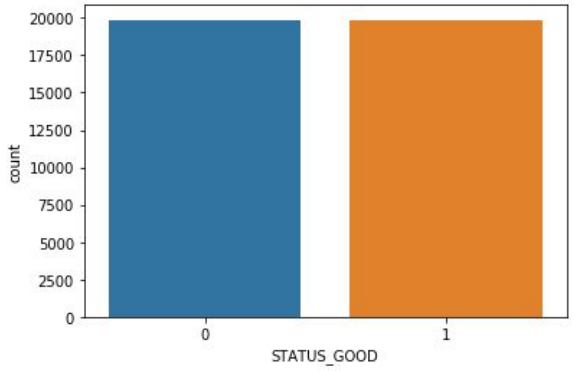
```
Relatório de Classificação:
              precision    recall  f1-score   support

           0     0.6813    0.7654    0.7209        81
           1     0.7361    0.6463    0.6883        82

    accuracy                         0.7055       163
   macro avg     0.7087    0.7059    0.7046       163
weighted avg     0.7089    0.7055    0.7045       163

Acurácia: 0.7055
```



Com undersample obteve-se uma acurária geral de 70,5%, porém, obteve sucesso parecido ao prever tanto bons pagadores (73%) quanto maus pagadores (68%)

Dataset resultante após o *oversample*:

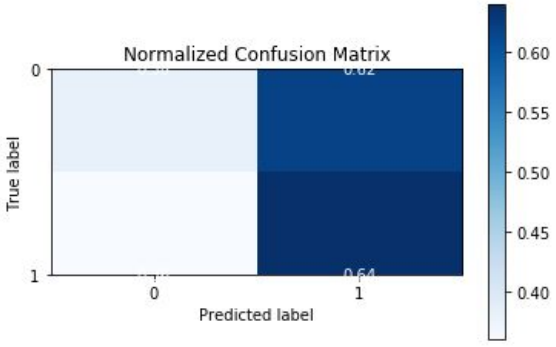
```
<matplotlib.axes._subplots.AxesSubplot at 0x2712d302108>
```



Avaliação da performance do dataset para a predição de bons pagadores após o *oversample*:

```
testa_modelo(df_oversampled)
```

```
Relatório de Classificação:
              precision    recall  f1-score   support

           0     0.5129    0.3815    0.4375      4965
           1     0.5076    0.6377    0.5653      4965

    accuracy                         0.5096      9930
   macro avg     0.5102    0.5096    0.5014      9930
weighted avg     0.5102    0.5096    0.5014      9930

Acurácia: 0.5096
```



Com oversample obteve-se uma acurária geral de 50,96%, sendo 51,29% para prever bons pagadores e 50,76% para prever maus pagadores. Isso já é um pouco melhor que o dataset desbalanceado que estava totalmente enviesado, mas ainda é muito próximo do mero acaso (50%)

Após o *undersample*, ainda foi realizada mais uma verificação para eliminar features que porventura tenham ficado sem variância com a eliminação de instâncias do dataset.

[115]:
```
#buscar por features que ficaram sem variância após o undersample
remover = df2.columns[(df2.var() == 0)]
df2.drop(columns=remover, inplace=True)
```

Uma vez resolvido o problema do balanceamento, foi procedido com a normalização dos valores do dataset. Utilizamos a biblioteca *sklearn* onde dentro dela importamos o *preprocessing,* neste utilizamos o método *StandardScaler,* com isso normalizamos as informações obtendo os dados preparados para métodos de aprendizado de máquina.

[119]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EMAIL	CNT_FAM_MEMBERS	CODE_GENDER_F	FLAG_OWN_CAR_Y	...	OCCUPATION_TYPE_Low-skill Laborers	OCC
0	-0.472275	1.466446	0.401041	-0.769055	-0.382602	-0.654654	-0.363874	-1.234183	0.647467	-0.688421	...	-0.096523	
1	1.169322	6.831288	1.109964	-0.757537	2.613680	-0.654654	-0.363874	1.178505	-1.544479	1.452600	...	-0.096523	
2	-0.472275	0.279991	0.277267	-0.761331	-0.382602	1.527525	-0.363874	-1.234183	0.647467	-0.688421	...	-0.096523	
3	-0.472275	-0.906465	-0.406687	-0.753416	2.613680	1.527525	-0.363874	-1.234183	0.647467	-0.688421	...	-0.096523	
4	-0.472275	0.331576	-0.301907	-0.805832	-0.382602	-0.654654	-0.363874	-0.027839	0.647467	-0.688421	...	-0.096523	
...	...	...	...	...	...	...	...	...	...	...	...	...	
645	-0.472275	-0.390615	-1.445957	1.311311	-0.382602	-0.654654	-0.363874	-1.234183	0.647467	-0.688421	...	-0.096523	
646	-0.472275	-0.390615	-1.445957	1.311311	-0.382602	-0.654654	-0.363874	-1.234183	0.647467	-0.688421	...	-0.096523	
647	-0.472275	-0.906465	-0.976258	1.311311	-0.382602	-0.654654	-0.363874	-1.234183	0.647467	1.452600	...	-0.096523	
648	-0.472275	-0.132689	-0.808524	1.311311	-0.382602	-0.654654	-0.363874	-1.234183	0.647467	-0.688421	...	-0.096523	
649	-0.472275	-0.132689	0.407656	-0.764645	-0.382602	-0.654654	-0.363874	-0.027839	0.647467	-0.688421	...	-0.096523	

650 rows × 47 columns

Agrupamento

A próxima etapa foi realizar os agrupamentos. Primeiramente utilizamos um agrupamento com o método *K Means.* Um método não-supervisionado da biblioteca *sklearn.cluster* que utiliza *clustering,* uma técnica que visa construir subgrupos para as amostras analisadas o qual possuam um certo grau de semelhança.
Para verificar a quantidade de clusters ideal, realizamos o cálculo do SSE e do Davies-Bouldin Score em um range de 2 a 100 clusters. Para cada um dos indicadores foi gerado um gráfico para auxiliar na avaliação da quantidade ideal de agrupamentos.

Gráfico do SSE de 2 a 100 grupos:

```
kmeansSSEEvaluating(df_scaled, 100)
```

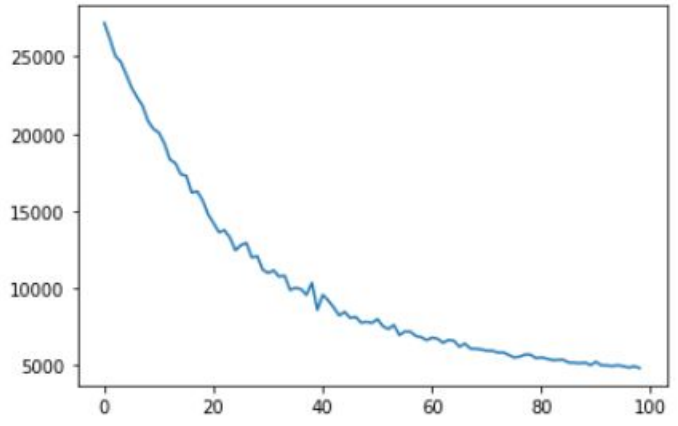
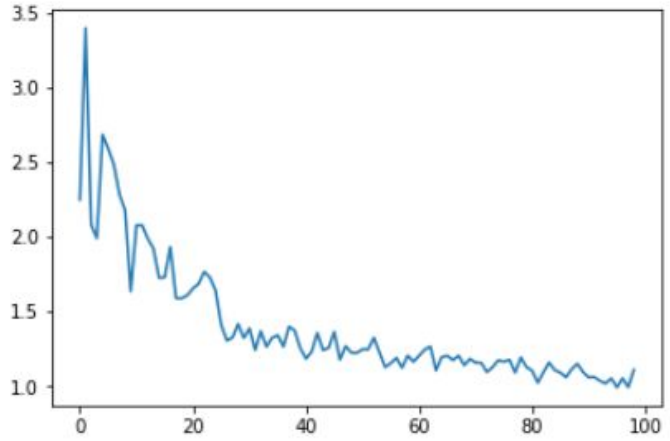


Gráfico do Davies-Bouldin Score de 2 a 100 grupos:

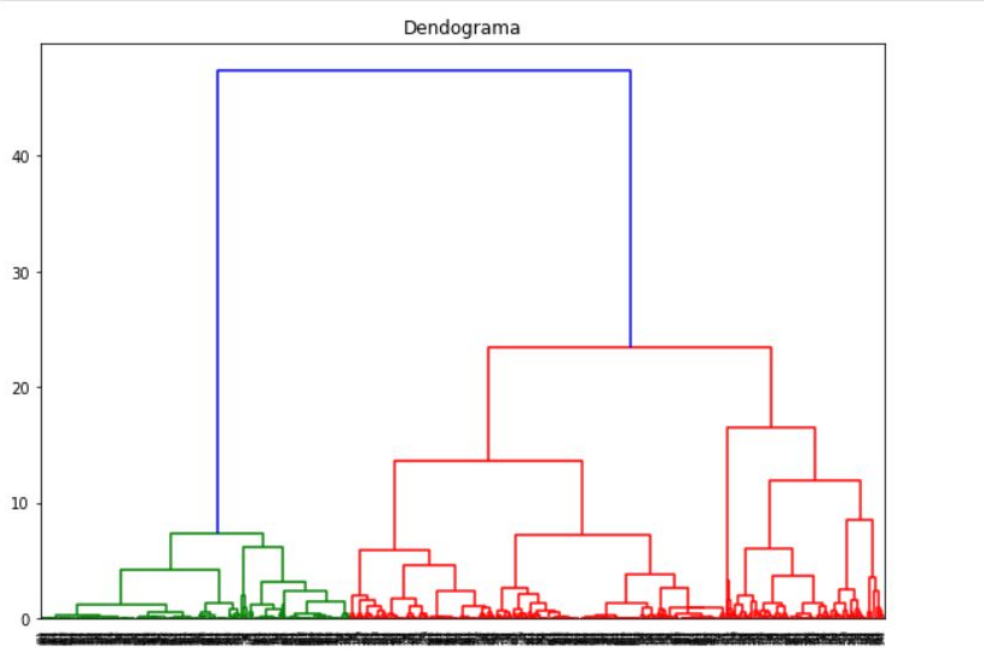
```
kmeansDBSEvaluating(df_scaled,100)
```



Analisando ambos os gráficos acima, chegamos a conclusão de que o número ideal fica em torno de 20 clusters para o método K-means.

Foram testados também métodos hierárquicos não-supervisionados para verificação dos agrupamentos, fazendo uso da biblioteca *sklearn.cluster.* Importamos o método AgglomerativeClustering onde ele iria iniciar com vários clusters pequenos e conforme sua execução via mesclando com o objetivo de fazer clusters maiores. Para validação utilizamos os dendogramas para descobrir o histórico de agrupamentos e descobrir o número ideal de clusters.

Para validação do agrupamento utilizado pelo K Means, foi utilizado os métodos da biblioteca *sklearn.metrics.cluster* o método *calinski_harabasz_score* e *davies_bouldin_score.*

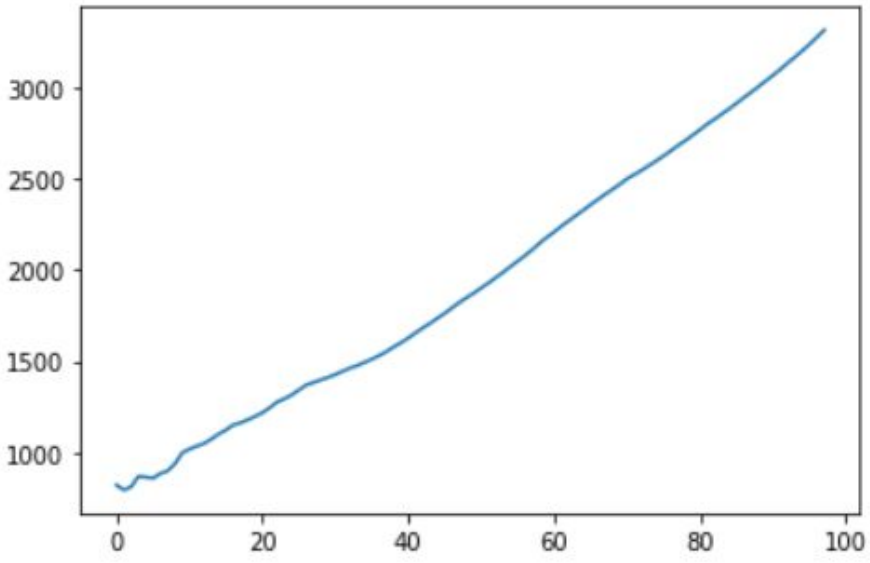


CALINSKI_HARABASZ_SCORE se baseia na ideia de que os agrupamentos que são bastante compactos e bem espaçados um dos outros são bons agrupamentos. Segundo o site [Cheat sheet for implementing 7 methods for selecting the optimal number of clusters in Python] em 2020, "*O índice é calculado dividindo a variância das somas dos quadrados das distâncias de objetos individuais ao centro do cluster pela soma dos quadrados da distância entre os centros do cluster. Quanto maior o valor do Índice Calinski-Harabasz, melhor o modelo de agrupamento*"
A fórmula deste método de validação é definida como:

$$CH_k = \frac{BCSM}{k-1} \cdot \frac{n-k}{WCSM}$$

[138]:
```
numGroups = 100
chbs = []
for n_groups in range (3,numGroups+1):

    f_labels = fcluster(h, t=n_groups, criterion='maxclust')
    score = chs(df_dendo,f_labels)
    chbs.append(score)
plt.plot(chbs)
```

[138]:
```
[<matplotlib.lines.Line2D at 0x271327f2d88>]
```

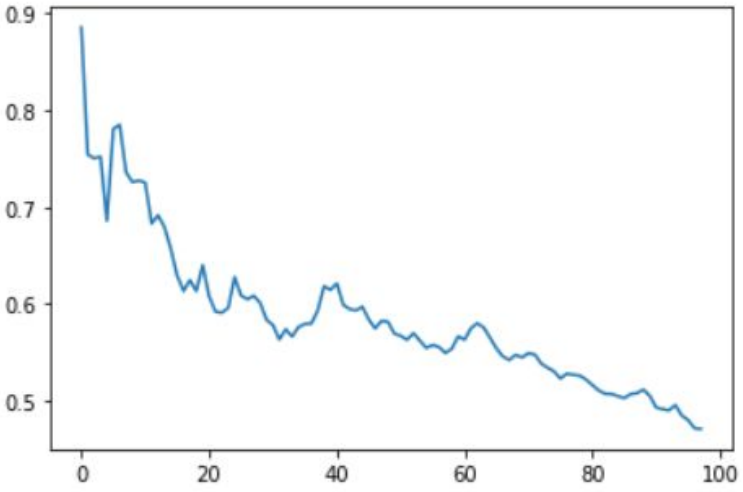


Para a validação *davies_bouldin_score.* Conforme o site da própria biblioteca, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html.
"*A pontuação é definida como a medida de similaridade média de cada cluster com seu cluster mais semelhante, onde a similaridade é a razão entre as distâncias dentro do cluster e entre as distâncias entre os cluster. Assim, os clusters mais distantes e menos dispersos resultarão em uma pontuação melhor.*
A pontuação mínima é zero, com valores mais baixos indicando melhor agrupamento."
A fórmula deste método de validação é definida como:

$$\mathrm{DB}(\mathcal{C}) = \frac{1}{k}\sum_{i=1}^{k} \max_{j \le k, j \ne i} D_{ij}, \; k = |\mathcal{C}|,$$

```
numGroups = 100
dbss = []
for n_groups in range (3,numGroups+1):

    f_labels = fcluster(h, t=n_groups, criterion='maxclust')
    score = dbs(df_dendo,f_labels)
    dbss.append(score)
plt.plot(dbss)
```

```
[<matplotlib.lines.Line2D at 0x2713279b088>]
```



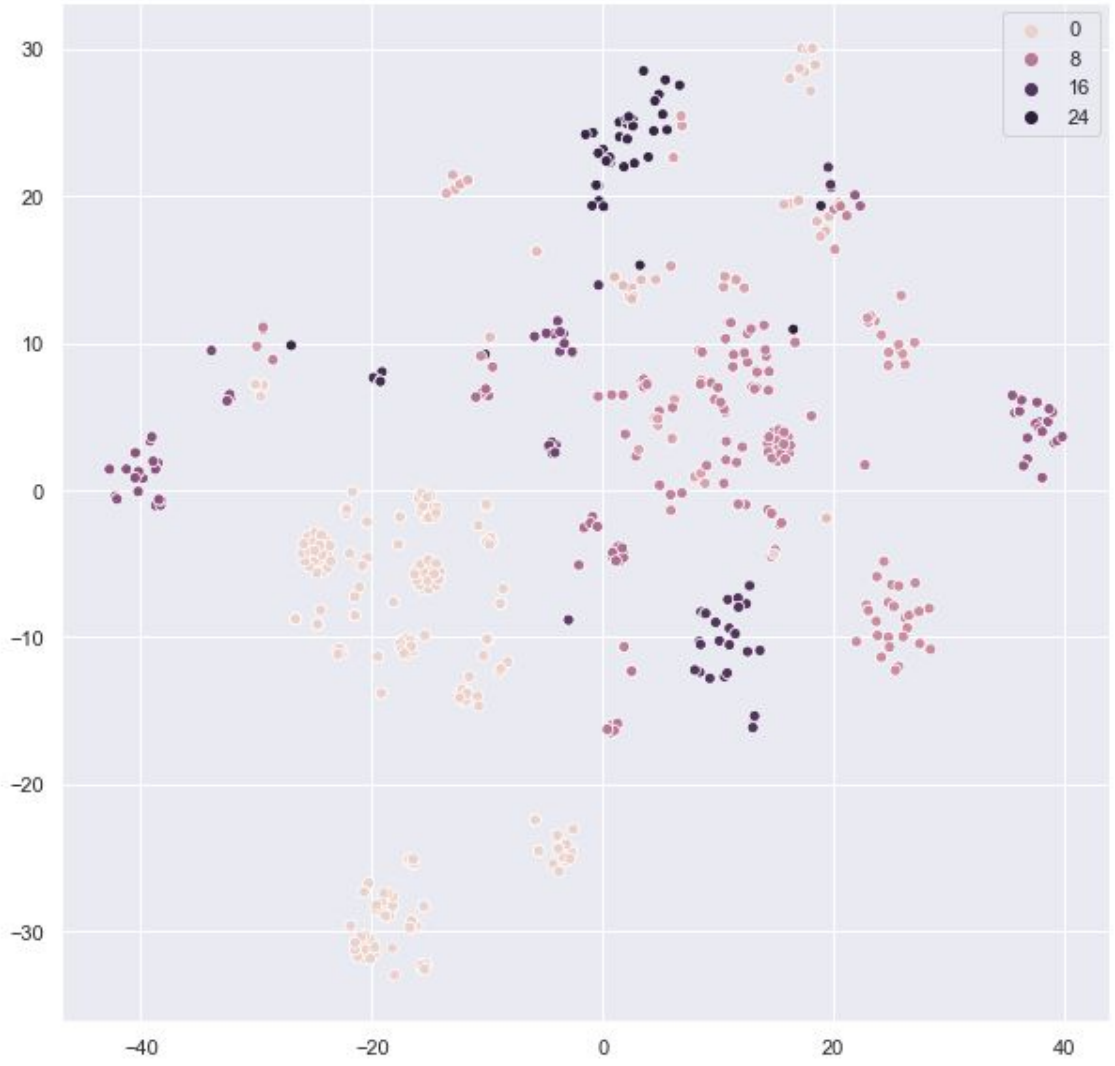
Resultados

A seguir serão apresentados os resultados encontrados nos agrupamentos k means e hierárquico.

K-means

Com o dataset normalizado, foi aplicado os algoritmos do k-means usando 20 clusters, valor este encontrado através da avaliação pelo SSE e Davies-Bouldin Score.

```
: <matplotlib.axes._subplots.AxesSubplot at 0x2712d1c4f48>
```

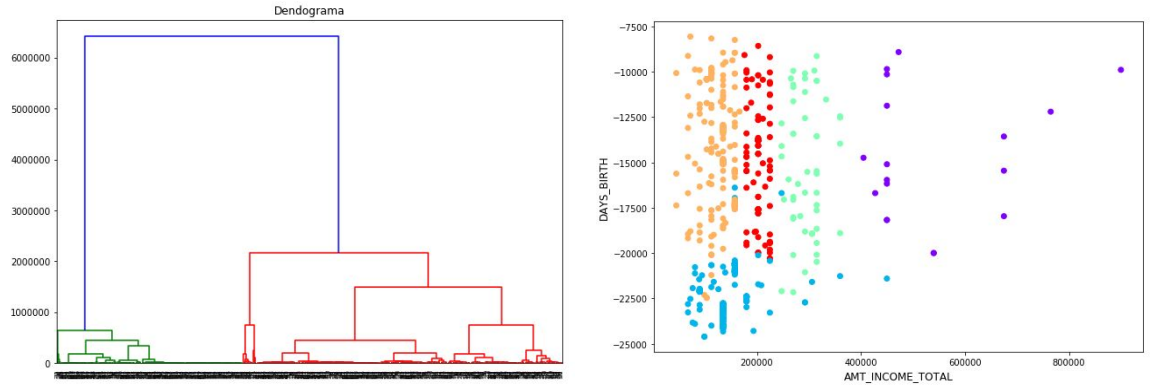


A análise da imagem acima mostra que os clusters estão dispostos, alguns próximos outros com uma distância maior.

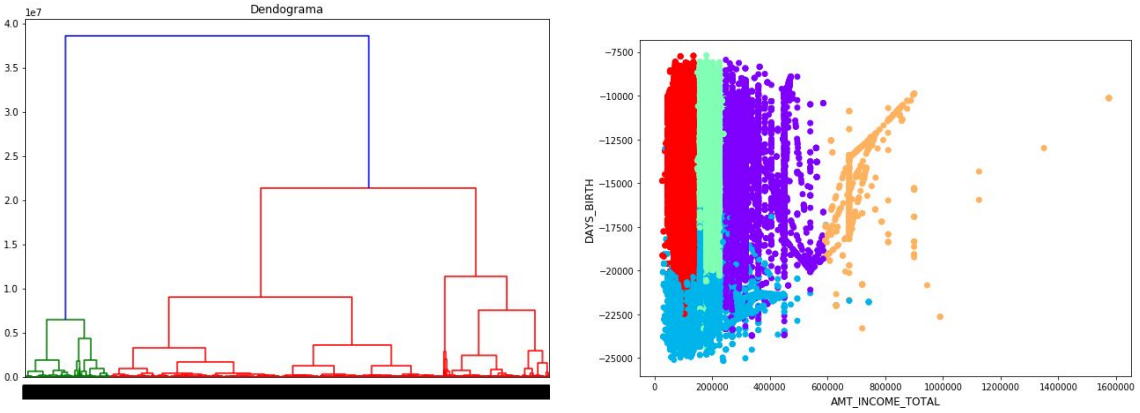
Algoritmos hierárquicos

Em uma primeira execução, os agrupamentos foram gerados com base no dataset balanceado via *undersample*, que foi o dataset utilizado para executar o k-means. Porém, como se pode notar nas imagens abaixo, a aplicação do algoritmo hierárquico sobre o dataset que sofreu *oversample* manteve a distribuição dos grupos com a vantagem de permitir uma melhor visualização.

Dendograma e Agglomerative Clustering aplicados ao dataset balanceado com *undersample*. Parâmetro *linkage* = *ward*:

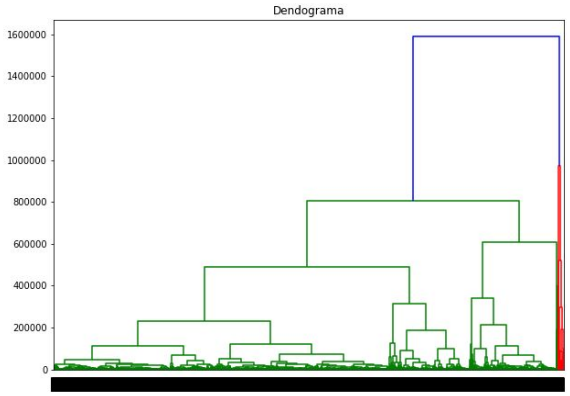


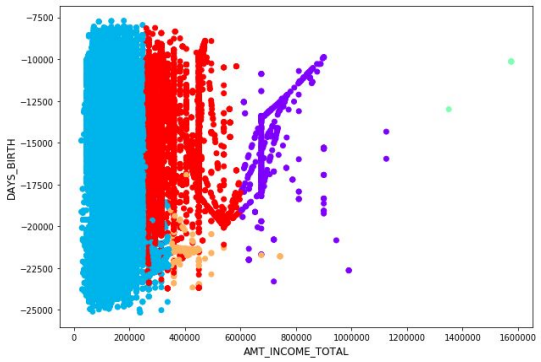
Dendograma e Agglomerative Clustering aplicados ao dataset balanceado com *oversample*. Parâmetro *linkage* = *ward*:



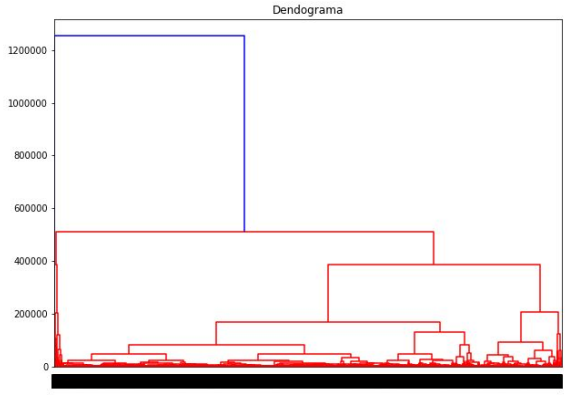
Posto isso, para as próximas análises de agrupamento usando algoritmo hierárquico, foi utilizado o dataset que foi balanceado por *oversample.*

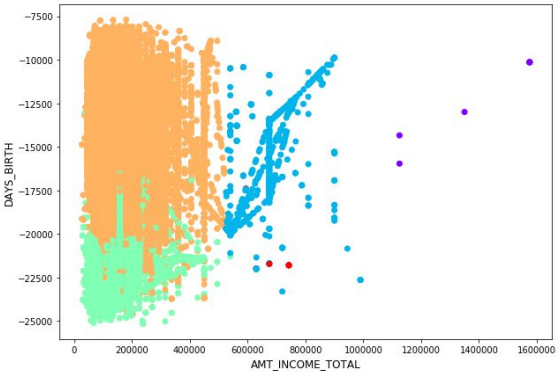
Dendograma e Agglomerative Clustering aplicados ao dataset balanceado com *oversample*. Parâmetro *linkage* = complete:



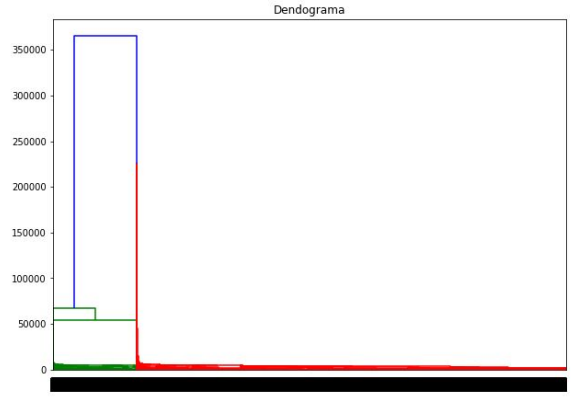


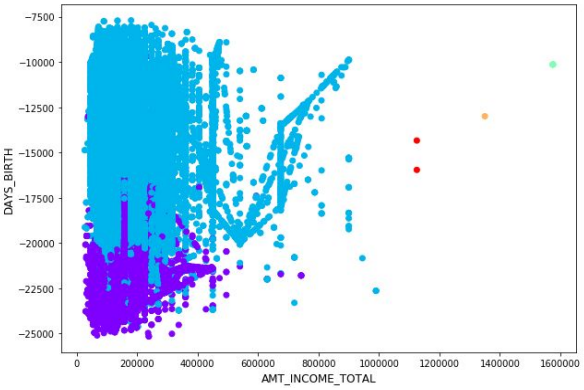
Dendograma e Agglomerative Clustering aplicados ao dataset balanceado com *oversample*. Parâmetro *linkage* = average:





Dendograma e Agglomerative Clustering aplicados ao dataset balanceado com *oversample*. Parâmetro *linkage* = single:





Comparando as técnicas de k means e com hierárquico pode-se identificar que nos hierárquicos ficou mais claro os clusters e outliers assim como ficou mais fácil a visualização gráfica por grupo.

Conclusão

O dataset escolhido proporcionou a experiência em diversas tarefas do dia a dia em um universo de machine learning, desde a análise exploratória, passando então para o pré-processamento para então aplicação dos algoritmos de clusterização k-means, hierárquico e agrupamento de dados.

As técnicas de aprendizagem não supervisionada tratadas durante a disciplina nos ajudaram a tratar dados não rotulados, explorar os dados e a identificar uma estrutura dentro deles. No universo do dataset que estudamos foi possível identificar features que se relacionam ao atributo target. Esse tipo de aplicação pode ajudar na identificação de possíveis maus pagadores, assim como conceder créditos a usuários com determinados perfis, que historicamente pagam as suas contas em dia.

É inegável o papel da IA e da machine learning no nosso mundo atual,m

Referências Bibliográficas

https://medium.com/data-hackers/como-lidar-com-dados-desbalanceados-em-problemas-de-classifica%C3%A7%C3%A3o-17c4d4357ef9

https://www.kaggle.com/rikdifos/credit-card-approval-prediction

https://medium.com/@saeedAR/smote-and-near-miss-in-python-machine-learning-in-imbalanced-datasets-b7976d9a7a79

https://towardsdatascience.com/machine-learning-algorithms-part-12-hierarchical-agglomerative-clustering-example-in-python-1e18e0075019

https://www.tandfonline.com/doi/abs/10.1080/03610927408827101

https://towardsdatascience.com/adasyn-adaptive-synthetic-sampling-method-for-imbalanced-data-602a3673ba16

https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

https://towardsdatascience.com/cheat-sheet-to-implementing-7-methods-for-selecting-optimal-number-of-clusters-in-python-898241e1d6ad

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html