Análise do Credit Card Fraud Detection Dataset

A análise a seguir se refere ao Credit Card Fraud Detection Dataset, um dataset público que contém informações sobre transações, normais e fraudulentas, feitas por cartões de créditos da Europa em setembro de 2013. Download feito no site https://www.kaggle.com/mlg-ulb/creditcardfraud. Utilizou-se o método de *machine learning – logistic regression* feito na linguagem de programação *python* para a análise do *dataset*.

O *dataset* contém mais de 280.000 transações, das quais apenas 492 são fraudulentas. Isso representa 0.172% de todas as transações, ou seja, temos um *dataset* desbalanceado, um possível problema para o modelo. O *dataset* possui 30 *features*, V1 a V28 (por razões de confidencialidade, não foi possível obter mais informações sobre as *features*), *Time* e *Amount*, o valor da transação. A *feature* Time contém o tempo decorrido entre a primeira transação e as demais. Não é uma informação relevante para análise, então foi desconsiderada. *Amount* é o quanto foi gasto na transação, portanto é uma *feature* possivelmente importante para a análise. V1 a V28 são os restantes das *features* que foram analisadas. A última coluna do *dataset* (*Class*) se refere se a transação foi ou não fraudulenta. Ela é o *target* do estudo.

Tabela 1 – Dados carregados no jupyter notebook

V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

Com os dados carregados no *jupyter notebook*, como visto na tabela 1 (V1 a V5 não apareceram na tabela 1, pois são muitos dados), foi separado em 2 variáveis os *inputs* e o *target*. I*nputs* foram as *features* V1 a V28 e *Amount*. Já o target foi Class.

Para uma melhor análise, os dados foram embaralhados randomicamente, já que eles coletados e armazenados numa mesma janela de tempo. No fim do estudo também foi feito um modelo sem o embaralhamento para comparação dos resultados.

Em seguida, foi feito o balanceamento do *dataset*, pois o número de transações fraudulentas em relação ao total é muito baixo e isso pode prejudicar o algoritmo do modelo, levando a interpretações e resultados errados.

O próximo passo foi separar o *dataset* em 80% dados de treino e 20% dados de teste que foram usados para checar a precisão final do modelo com dados novos.

Tabela 2 – Valores dos coeficientes e *intercept* - dados randomizados

Feature Name	***Coefficient***
Intercept	0,0630
V24	0,2259
V28	0,1673
V25	0,1204
V22	0,0880
V19	0,0823
V14	0,0627
V11	0,0597
V20	0,0553
V9	0,0540
V3	0,0370
V21	0,0329
V4	0,0264
V5	0,0139
V26	0,0007
Amount	-0,0003
V23	-0,0045
V6	-0,0058
V15	-0,0130
V10	-0,0277
V17	-0,0709
V16	-0,0880
V27	-0,1035
V8	-0,1258
V18	-0,1348
V12	-0,1349
V13	-0,1352
V1	-0,1415
V2	-0,1535
V7	-0,2394

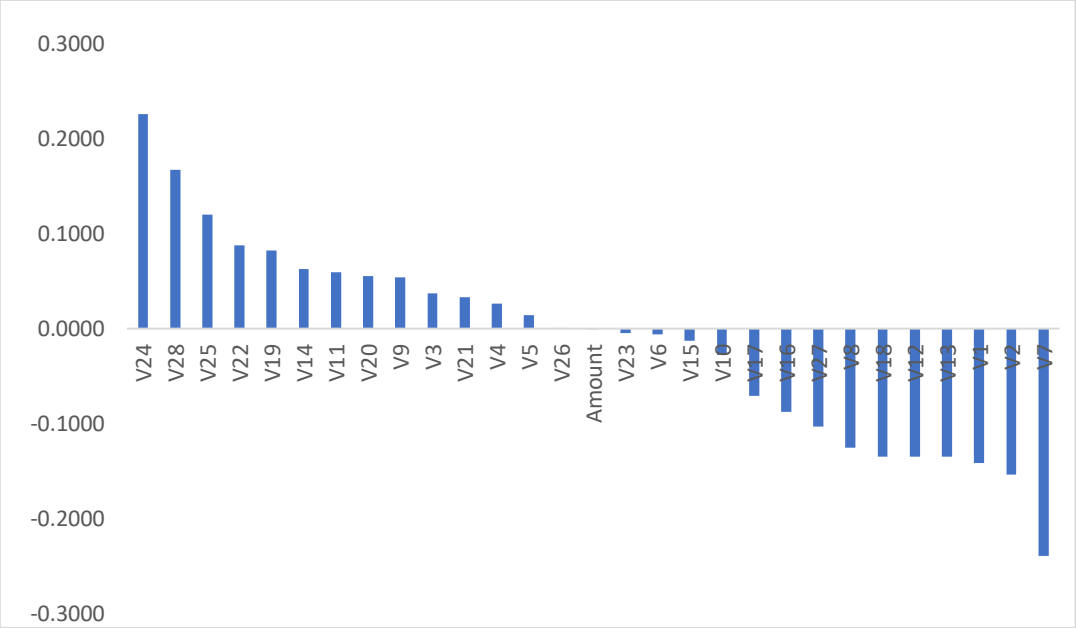

Figura 1 - Valores dos coeficientes - dados randomizados

Na tabela 2, temos os valores dos coeficientes para o modelo *logistic regression* e na figura 1 a representação visual dos valores. Pode-se perceber que o valor de *Amount* é próximo de zero e não é impactante para o modelo, ou seja, na prática, o valor das transações não é um fator relevante para classificar se a transação é fraudulenta ou não. Pode ser uma transação de R$5,00 ou R$10.000,00, não é importante. Outras *features* que não são relevantes são V26, V23 e V6. Não é possível verificar a razão desse comportamento, pois não foram fornecidas as informações sobre o que representa cada *feature*.

Tabela 3 – Valores da precisão do modelo – dados randomizados

Precisão	
Dados Treino	57,31%
Dados Teste	50,25%

Os resultados obtidos da precisão do modelo podem ser vistos na tabela 3. A diferença obtida com os dados de treino e teste são esperados. Um possível motivo para que a precisão do modelo não tenha sido maior, deve-se ao fato que o *dataset* é grande, mas o número de transações fraudulentas é muito baixo (0,147% do total). O *dataset* original contém mais de 280 mil colunas de dados e, após o balanceamento, foi obtido um *dataset* com mais de 900 colunas, um número pequeno em relação ao original, o que compromete a performance do algoritmo. Portanto, o baixo volume de casos positivos pode ter contribuído para a precisão do modelo não ter sido maior. Com um *dataset* que contenha mais casos positivos em relação ao total, possivelmente a precisão do modelo será maior.

Tabela 4 – Comparação entre dados randomizados e não randomizados

	Dados Randomizados	Dados não randomizados
Precisão (treino)	57,31%	96,57%
Precisão (teste)	50,25%	95,43%

Foi feito um segundo estudo com os dados, depois do balanceamento, não randomizados, a fim de verificar possíveis ganhos em precisão. Foi verificado que, sem randomizar os dados, temos uma precisão muito maior, como visto na tabela 4, tanto com o *set* de treino, como com o *set* de teste. Possivelmente o algoritmo detectou um padrão nos dados para obter tamanha precisão. Temos que lembrar que os dados foram coletados e dispostos em ordem cronológica, o que pode ter formado o padrão e causado um *overfitting*, resultando a precisão de 96,57 % e invalidando o modelo.