Nanodegree Engenheiro de Machine Learning

Detectar transações fraudulentas de cartão de crédito

Talita Shiguemoto
17 de novembro de 2018

I. Definição

Visão Geral do Projeto

Segundo a *Konduto*[1], a cada cinco segundos uma tentativa de fraude por cartão de crédito clonado ocorre em *e-commerces* brasileiros. De acordo mesma empresa, no ano passado o Brasil sofreu cerca de 6 milhões de compras fraudulentas. Comparado a outros, um estudo realizado em 2016 pela *Business Insider Intelligence*[2] mostrou o Brasil em segundo lugar no ranking de maior porcentagem de consumidores expostos a transações fraudulentas via cartão de crédito nos últimos cinco anos, como podemos observar na Figura 2.

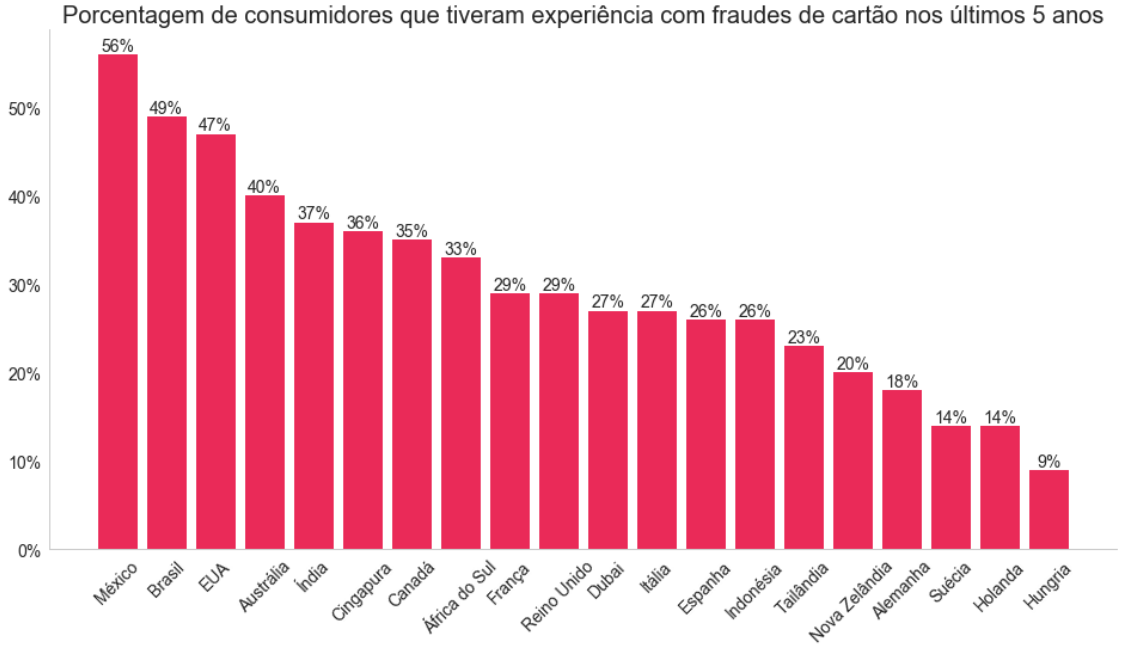

Figura 1 - Ranking Países com transações fraudulentas

[1] https://www.konduto.com/
[2] https://www.businessinsider.com/the-us-has-the-third-highest-card-fraud-rate-in-the-world-2016-7

Este projeto iniciou-se para entender como combater este cenário nacional, tendo como objetivo criar um algoritmo de detecção das transações com fraudes baseadas no conjunto de dados reais de cartões europeus, tal base foi retirada de uma plataforma focada em ciência de dados, o *Kaggle*[3].

Descrição do problema

Muitas transações por cartão de crédito são fraudulentas, o objetivo deste projeto é detectá-las baseado em um conjunto de dados histórico que já possui tal classificação (se foi fraude ou não). Cada transação possui algumas variáveis e com base nelas poderemos identificar quais atributos possuem maior influência para desvendar quais são fraudes ou não. Devido esse histórico com características o modelo criado será baseado em uma técnica de *Machine* Learning chamada aprendizagem supervisionada e poderá ser utilizado para futuras detecções de fraudes.

O primeiro passo será pré-processar os dados para lidarmos com o desbalanceamento que, em outras palavras, é o fato de que normalmente em transações por cartão de crédito a taxa de fraudes é muito menor que a taxa de transações verdadeiras. Portanto, iremos utilizar uma técnica de subamostragem[4] para resolver este impasse. Feito isso, será testado os algoritmos de Métodos de Ensemble (Random Forest, Gradient Boosting, XGBoost), Support Vector Machines e K-Nearest Neighbors, verificando qual deles terá melhor performance e resultados. O modelo final será escolhido embasado por algumas métricas (veja em **Métricas de avaliação**) e terá seus hiperparâmetros otimizados pelo *GridSearch*[5].

Métricas de avaliação

Devido nosso conjunto ser desbalanceado a acurácia não é a melhor métrica para ser usada na avaliação do modelo. Para este estudo utilizaremos principalmente o *AUC ROC* para verificar a qualidade do algoritmo.

O *recall,* também chamado de ***True Positive Rate (TPR)***, é a proporção entre os Verdadeiros Positivos sobre todas os positivos classificados corretamente e os falsos negativos. Matematicamente representado por:

[3] https://www.kaggle.com/mlg-ulb/creditcardfraud/home
[4] http://www.teses.usp.br/teses/disponiveis/55/55134/tde-06012016-145045/publico/VictorHugoBarella_dissertacao_revisada.pdf
[5] http://scikit-learn.org/stable/modules/grid_search.html

$$Recall = \frac{TP}{TP + FN}$$

Para entender o AUC é necessário compreender primeiro a curva do ***Receiver Operating Characteristic (ROC)***[6], que é um gráfico que mostra o desempenho de um modelo de classificação em todos os limites de classificação. Esta curva mostra dois parâmetros: o TPR e o False Positive Rate (FPR).

$$FPR = \frac{FP}{TP + TN}$$

Uma curva ROC traça TPR vs. FPR em diferentes limiares de classificação. Diminuir o limiar de classificação classifica mais itens como positivos, aumentando assim tanto os falsos positivos quanto os verdadeiros positivos. Em outras palavras, um modelo prediz a probabilidade de uma classe ser 1 ou 0, usando essas probabilidades é possível plotar um gráfico de distribuição como na Figura 2, sendo a curva vermelha representando 0 e a verde para 1[7], sendo 0.5 o limite entre as duas classes.

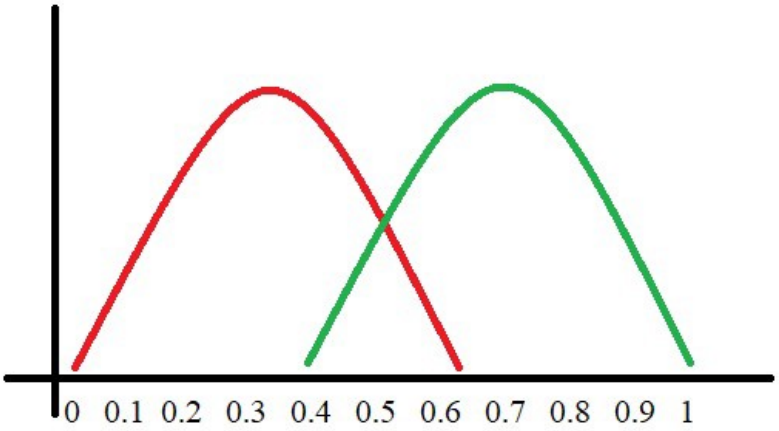

Figura 2 - ROC

Todos os valores positivos acima do limite (maiores que 0.5) serão ***True Positives (TP)***, e todos os valores negativos acima do limite serão ***False Positives (FP)***, uma vez que foram classificados incorretamente como positivos. Abaixo do limite, todos os valores negativos serão ***True Negatives (TN)*** e os positivos ***False Negatives (FN)***, uma vez que foram classificados incorretamente como negativos. Esse conceito é melhor demonstrado na Figura 3.

[6] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
[7] https://medium.com/greyatom/lets-learn-about-auc-roc-curve-4a94b4d88152

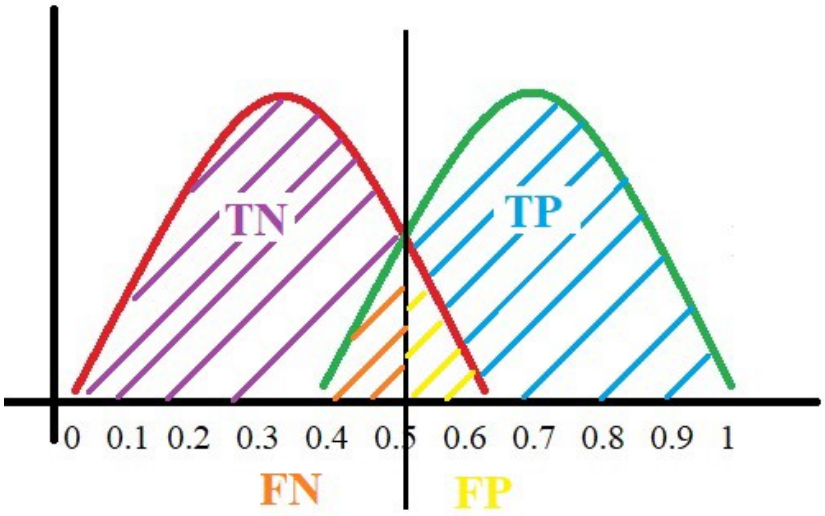

Figura 3 - TN, TP, FN, FP

A curva AUC mede toda a área bidimensional por baixo de toda curva ROC. AUC fornece uma medida agregada de desempenho em todos os possíveis limites de classificação. Uma maneira de interpretar AUC é como a probabilidade de o modelo classificar um exemplo positivo aleatório mais do que um exemplo negativo aleatório. Um modelo cujas previsões são 100% erradas tem uma AUC de 0,0; enquanto um cujas previsões são 100% corretas tem uma AUC de 1,0. Conforme Figura 4.

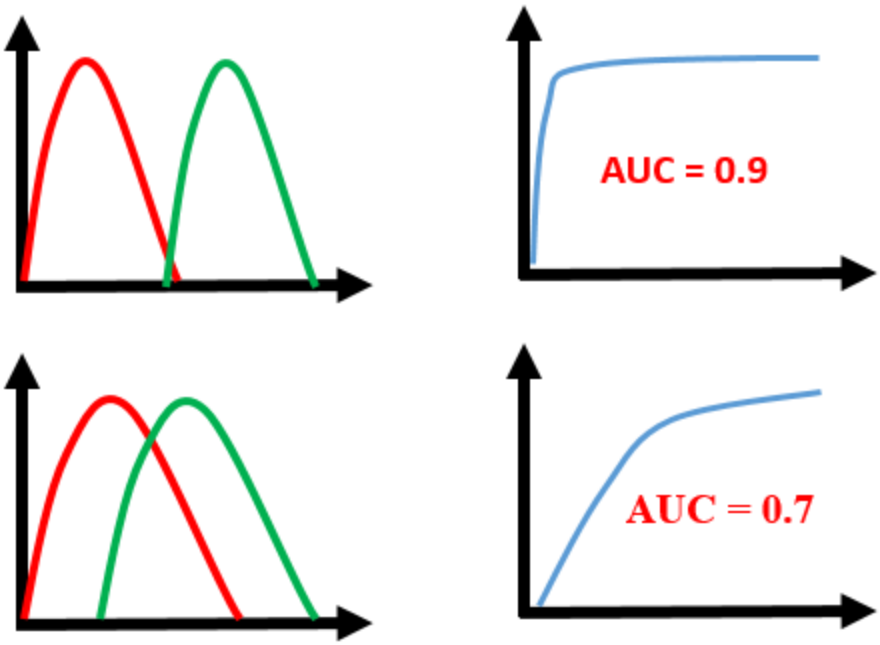

Figura 4 - ROC AUC

Outra métrica utilizada para escolha do melhor modelo será a utilização do *Classification Report*[8] do *sklearn*, trata-se de uma tabela de reporte com as principais métricas de avaliação para cada classe, sendo elas: *recall, precision* e *f1_score*[9].

O *precision*[10] é a proporção entre os Verdadeiros Positivos sobre todas os positivos, sendo eles classificados corretamente ou não. Matematicamente representado por:

$$Precision = \frac{TP}{TP + FP}$$

O *f1_score* é uma medida que considera ambos: *precision* e *recall*, sendo sua fórmula expressada por:

$$F\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

II. Análise

Exploração dos Dados

O conjunto de dados é referente a transações de usuários de cartões europeus em setembro de 2013, obtidos no *Kaggle*. Possuindo 284.807 linhas e 31 *features*, das quais 28 delas são variáveis dependentes que devido a confidencialidade são resultados de transformações de PCA. Podemos verificar em detalhes cada *feature* na Tabela 1.

Nome	Descrição	Tipo	Valores
Time	Segundos transcorridos entre cada transação	Quantitativo	-
V1-V28	Variáveis dependentes anônimas devido confidencialidade	Quantitativo	-
Amount	Montante da Transação	Quantitativo	-
Class	Variável resposta. Se a transação foi fraudulenta	Qualitativo	0 (falso), 1 (verdadeiro)

Tabela 1 - Descrição do Conjunto de Dados

[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
[9] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
[10] https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

Não há nenhum *missing values*[11] no conjunto de dados, contudo os dados estão desbalanceados como podemos observar na Figura 5, sendo que a variável *target Class* possui valores binários, 0 para transações normais e 1 para fraudulentas, na qual a segunda representa 0,17% dos dados.

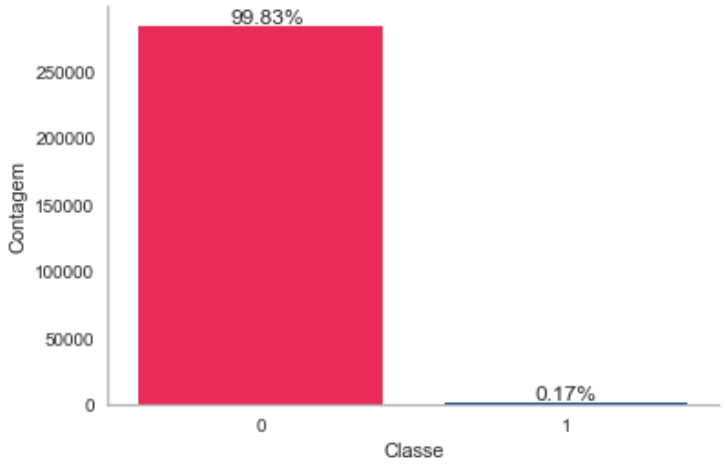

Figura 5 - Dados desbalanceados

Devido as *features* anônimas já serem fruto de um Pré-Processamento, uma transformação via PCA, foi verificado como se comporta a distribuição da variável *Amount*, na qual podemos ver na Figura 6 que ela possui uma distribuição assimétrica à direita *(positive skewed)*.

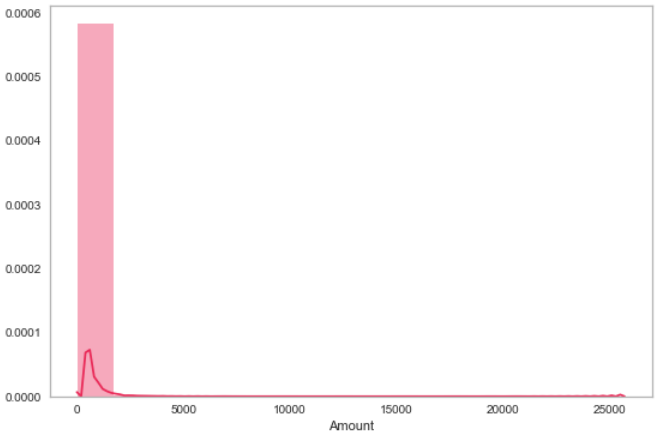

Figura 6 - Distribuição Assimétrica de Amount

[11] https://en.wikipedia.org/wiki/Missing_data

Visualização Exploratória

Nosso conjunto de dados para preservar a confidencialidade possui 28 variáveis transformadas via PCA, sem estarem com seus verdadeiros rótulos, devido a isso a criação de hipóteses sobre elas torna-se um desafio um pouco maior.

A suposição é que devido esta transformação elas não possuam correlação entre si, uma vez que PCA cria uma ortogonalidade entre essas *features*, tal fato é comprovado no Mapa de Calor na Figura 7. As variáveis que conhecemos são o Montante (*Amount*), Tempo em segundos (*Time*) e a Classe (*Class*). Na Figura 8 podemos avaliar como se comportam essas três *features* entre si, não sendo possível notar nenhum padrão.

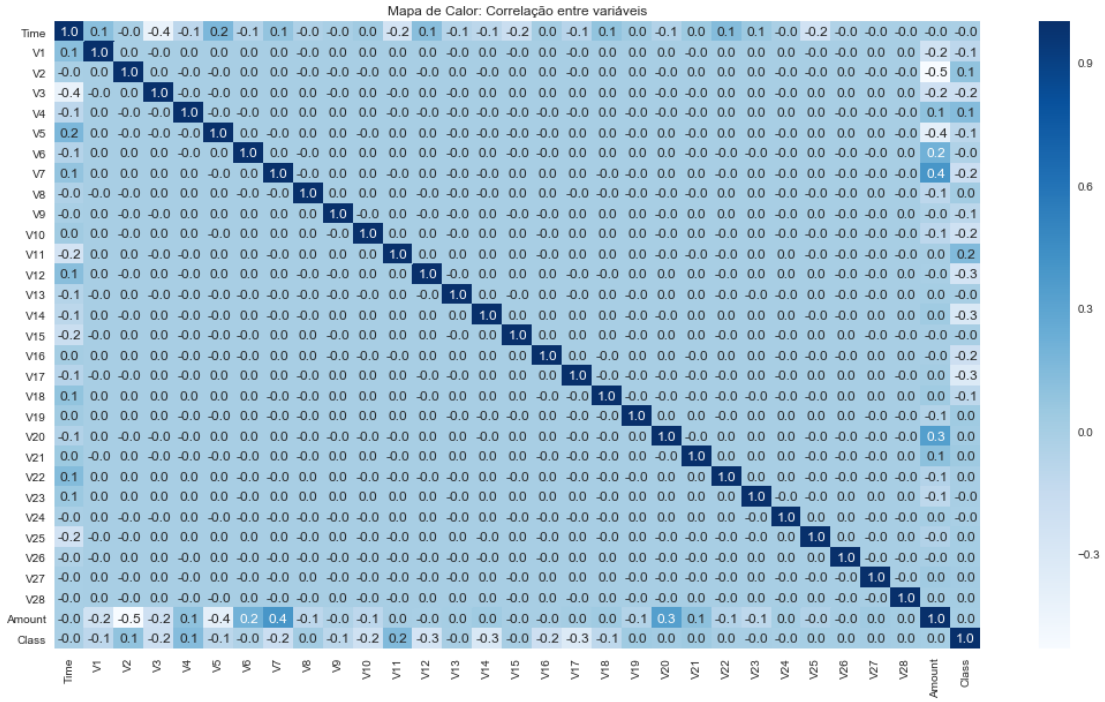

Figura 7 - Mapa de Calor: Correlação entre as variáveis

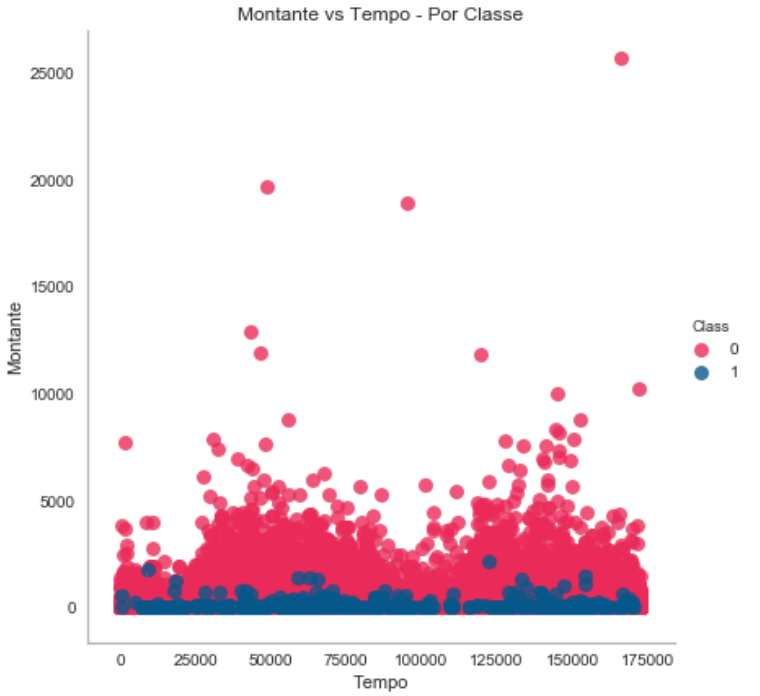

Figura 8 - Montante vs Tempo por Classe

Algoritmos e Técnicas

Neste projeto testaremos uma série de algoritmos da biblioteca do *sklearn,* na intenção de optar pelo o que obtiver um melhor resultado nas análises. Os modelos utilizados neste estudo são:

- **Support Vector Machines (SVM)**: utiliza uma técnica chamada de *kernel* para transformar os dados e encontra um limite ideal entre as saídas possíveis. Possui uma alta acuracidade, não é sensível a não linearidade dos dados, baixo risco de *overfitting* usando *kernels* corretos, trabalha bem com dados dimensionais elevados.[12]

- **Regressão Logística (RL)**: estima a probabilidade associada à ocorrência de um determinado evento devido algumas *features*. Possui alto grau de confiabilidade e não é necessário supor normalidade multicolinearidade. [13]

[12] https://statinfer.com/204-6-8-svm-advantages-disadvantages-applications
[13] https://edisciplinas.usp.br/pluginfile.php/3769787/mod_resource/content/1/09_RegressaoLogistica.pdf

- **Random Forest (RF)**: um método ensemble utilizado para construir um modelo baseado em múltiplas Decision Trees durante a fase de treino. Possui baixo risco de *overfitting*, ele costuma ser rápido para treinar.[14]

- **XGBoost (XGB)**: é uma implementação avançada do Gradient Boosting, um método ensemble que tenta minimizar o Erro Médio Quadrático até que a soma dos resíduos seja próxima de 0 (ou mínimo) e os valores previstos estejam suficientemente próximos dos valores reais.**Error! Bookmark not defined.** **Error! Bookmark not defined.** O XGB é mais rápido e com alto poder preditivo. Possui uma variedade de regularizações que reduzem o *overfitting* e melhoram o desempenho geral. **Error! Bookmark not defined.**

As técnicas utilizadas neste projeto são:

- **GridSearch**: otimiza os hiperparametros dos modelos para encontrar o modelo mais otimizado possível.[15]

- **Under-sampling**: retira da modelagem linhas da classe majoritária, para evitar desbalanceamento.[16]

- **Over-sampling**: replica linhas da classe minoritária ou gera dados sintéticos, para evitar desbalanceamento.[16]

- **Synthetic Minority Over-Sampling Technique (SMOTE):** Utilizado para sintetizar os dados no *over-sampling*.[17] Tal técnica será implementada junto com *Tomek links* para evitar o aumento de variância causada por algumas técnicas de geração de dados sintéticos.

- **Transformação em Log**: utilizado para normalizar os dados com distribuição assimétrica.

- **MinMaxScaler**: padroniza as *features* na mesma escala.[18]

[14] https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd
[15] http://scikit-learn.org/stable/modules/grid_search.html
[16] http://www.scielo.br/pdf/ca/v22n5/v22n5a02.pdf
[17] https://imbalanced-learn.org/en/stable/generated/imblearn.over_sampling.SMOTE.html
[18] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Benchmark

Utilizando o mesmo conjunto de dados no estudo[19] mais votado no *Kaggle*, o autor utilizou uma Regressão Logística para criar o modelo que obteve uma acurácia e recall próximo de 93% e, conforme a Figura 9, obteve a *Area Under the ROC Curve (AUC)* em aproximadamente 0,95. Para contornar o problema de desbalanceamento o autor fez uma submamostragem e executou tanto o treino do algoritmo como teste nos dados subamostrados, obtendo o 0,95 de *ROC AUC*.

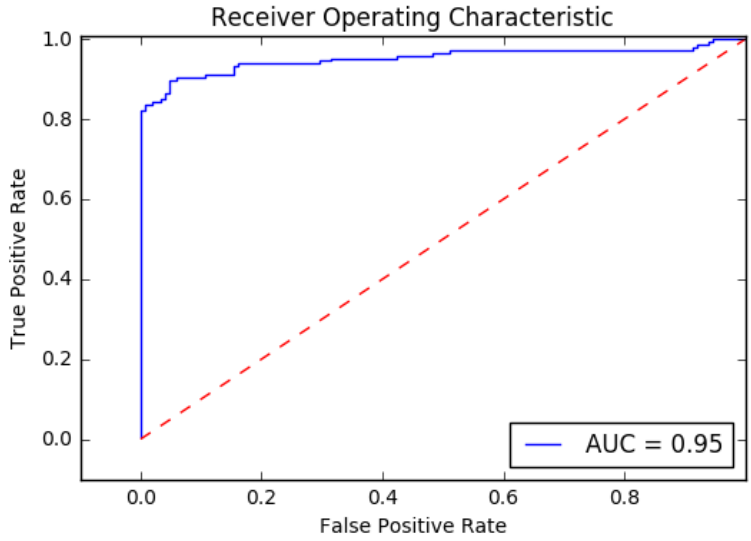

Figura 9 - AUC Benchmark

III. Metodologia

Pré-Processamento dos dados

Devido a *feature Amount* ter uma distribuição assimétrica, com alguns dados sendo 0, efetuei uma transformação em log(x+1) e então passei para o *Feature Scaling*, que consiste em reescalonar os dados para obter uma distribuição com média de zero e desvio padrão de um. A técnica utilizada para esta etapa foi o ***MinMaxScaler*** do sklearn, na coluna *feature* pelo log, tornando os dados em uma escala de 0 a 1. Foi criado um

[19] https://www.kaggle.com/joparga3/in-depth-skewed-data-classif-93-recall-acc-now

novo *dataset* para armazenar os dados originais mais o transformado e foi retirado as *features* Time e *Amount* original.

Uma vez que temos os dados prontos para analisar, faz-se necessário dividir os dados entre conjuntos de treinamento e de teste, em uma proporção de 70% para treinamento e 30% para teste. Por fim, para tratar o desbalanceamento dos dados foi necessário reamostrá-los, utiliznado somente os dados de treinamento, nesta etapa foi criada seis novos conjuntos de dados com diferentes técnicas de reamostragem, como podemos observar na Tabela 2.

Dataframe	Técnica utilizada	% Class 1	% Class 0
df_undersample50	UnderSampling aleatório	50,0%	50,0%
df_undersample33	UnderSampling aleatório	33,3%	66,7%
df_oversample50	OverSampling aleatório	50,0%	50,0%
df_oversample33	OverSampling aleatório	33,3%	66,7%
df_Smote50	SMOTE com Tomek links	50,0%	50,0%
df_Smote33	SMOTE com Tomek links	33,3%	66,7%

Tabela 2 - Técnicas de Reamostragem

Implementação

Nesta etapa foi criada uma função para aplicar um algoritmo classificador sem nenhum parâmetro otimizado e já mostrar os resultados das métricas citadas no capítulo **Métricas de avaliação** e o tempo que o algoritmo usa para treinar e prever os dados, baseados nos *datasets* resultantes de reamostragem. O intuito desta etapa foi verificar qual o algoritmo e técnica de reamostragem que tem melhor performance nos dados de treinamento e teste para encontrarmos o melhor modelo.

Na Tabela 3 estão todos os algoritmos separados por técnicas de reamostragem, mostrando o desempenho de cada modelo ordenado pelo valor mais alto em ROC AUC, sendo a métrica como principal influenciadora na decisão da escolha do modelo, seguido por *f1_score* para classe 1 e depois classe 0.

Podemos concluir observando a Tabela 3 que o algoritmo RL unido com a técnica de reamostragem *Smote with Tomek Links* com balanceamento de classes iguais, obteve a melhor pontuação de *ROC AUC* (0,947), contudo o *f1_*score para classe 1 foi muito baixo, sendo 0,110, portanto o algoritmo escolhido foi o que desempenhou os melhores resultados balanceados entre as duas métricas, sendo o algoritmo RF com a mesma técnica de reamostragem do modelo acima, obtendo o *ROC AUC* em 0,905 e *f1_score* em 0,850, com o tempo de treinamento e predição dos dados mediano.

Model	Dataset	roc_auc_test	f1_score 0	f1_score 1	recall 0	recall 1	precision 0	precision 1	time_pred	time_train
RL	df_Smote50	0,947	0,990	0,110	0,980	0,920	1,000	0,060	0,052	6,245
RL	df_undersample50	0,946	0,980	0,060	0,950	0,940	1,000	0,030	0,000	0,001
RL	df_oversample50	0,944	0,990	0,120	0,980	0,910	1,000	0,060	0,047	5,673
RL	df_Smote33	0,940	0,990	0,210	0,990	0,890	1,000	0,120	0,090	4,791
RF	df_undersample50	0,938	0,990	0,120	0,980	0,900	1,000	0,070	0,160	0,058
RL	df_oversample33	0,937	0,990	0,220	0,990	0,880	1,000	0,130	0,047	4,424
XGB	df_Smote33	0,937	1,000	0,400	1,000	0,880	1,000	0,260	1,563	146,169
XGB	df_undersample33	0,936	0,990	0,210	0,990	0,880	1,000	0,120	0,414	0,369
RL	df_undersample33	0,936	0,990	0,140	0,980	0,890	1,000	0,080	0,007	0,012
XGB	df_Smote50	0,934	1,000	0,250	0,990	0,880	1,000	0,150	1,955	173,514
XGB	df_undersample50	0,933	0,980	0,070	0,960	0,900	1,000	0,040	0,372	0,199
SVM	df_undersample33	0,933	0,990	0,100	0,970	0,890	1,000	0,060	2,449	0,103
RF	df_undersample33	0,931	1,000	0,270	0,990	0,870	1,000	0,160	0,193	0,069
SVM	df_undersample50	0,928	0,970	0,050	0,940	0,920	1,000	0,020	1,801	0,046
XGB	df_oversample33	0,928	1,000	0,560	1,000	0,860	1,000	0,420	1,452	86,181
XGB	df_oversample50	0,927	1,000	0,420	1,000	0,860	1,000	0,280	2,135	116,094
RF	df_Smote50	0,905	1,000	0,850	1,000	0,810	1,000	0,890	1,005	51,334
RF	df_Smote33	0,901	1,000	0,850	1,000	0,800	1,000	0,910	2,262	38,401
RF	df_oversample50	0,884	1,000	0,850	1,000	0,770	1,000	0,950	0,893	23,328
RF	df_oversample33	0,878	1,000	0,840	1,000	0,760	1,000	0,950	0,642	20,336
SVM	df_Smote33	0,850	1,000	0,550	1,000	0,700	1,000	0,460	162,000	1,071
SVM	df_Smote50	0,849	1,000	0,500	1,000	0,700	1,000	0,390	210,413	2,898
SVM	df_oversample50	0,846	1,000	0,540	1,000	0,690	1,000	0,450	170,741	13,543
SVM	df_oversample33	0,846	1,000	0,540	1,000	0,690	1,000	0,450	136,159	3,117

Tabela 3 - Performance dos Modelos

Refinamento

Tendo o modelo e técnica de reamostragem definidas o passo seguinte foi a otimização do algoritmo, para isso foi utilizado a biblioteca *GridSearch* do *sklearn*. Aqui se faz importante ressaltar que uma vez que nossa métrica de avaliação será *ROC AUC*, tendo um algoritmo escolhido, utilizaremos a predição dele com probabilidade, que no caso do XBG é *predict_proba,* isso porque como explicado no capítulo de **Métricas de avaliação**, a curva ROC AUC é feita variando o *threshold* (limite) para encontrar as taxas de falso positivo / negativo, calculando as probabilidades para cada classe.

O resultado da *ROC AUC* com *predict_proba* no modelo sem refinamento foi 0.9439, com os seguintes hiperparametros:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini'
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None
            oob_score=False, random_state=96, verbose=0, warm_start=False)
```

Para melhor ainda mais o desempenho do modelo foi utilizado o *GridSearch* otimizando os hiperparâmetros de acordo com a Tabela 4.

Parâmetro	Descrição	Valores Testados	Melhor Valor
min_samples_split	O número mínimo de amostras necessárias para dividir um nó interno	2, 3 e 5	2
min_samples_leaf	O número mínimo de amostras necessárias para estar em um nó de folha.	4, 5 e 6	5
max_depth	Maior profundidade da árvore	5, 10, 15 e 20	10
max_features	O número de features para ser usado no modelo	auto e sqrt	auto
n_estimators	O número de árvores na Forest	200, 288, 377, 466, 555, 644, 733, 822, 911 e 1000	555

Tabela 4 - Descrição de Hiperparâmetros

IV. Resultados

Avaliação e Validação do Modelo

Com a otimização do modelo feito no passo anterior, o resultado final da *ROC AUC* com *predict_proba* no modelo otimizado foi 0.9823, podendo ser verificado na Figura 10, com os seguintes hiperparametros:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini'
            max_depth=10, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=822, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

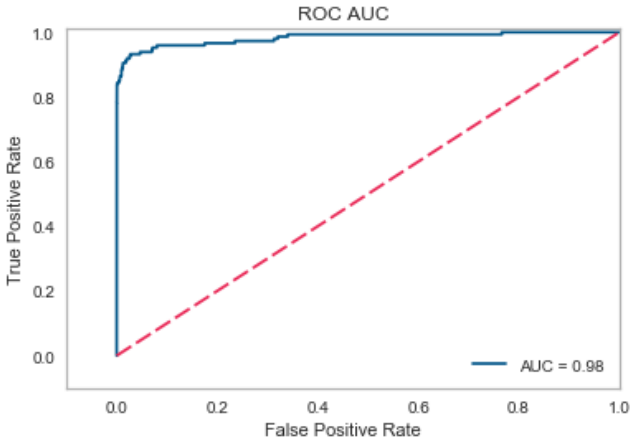

Figura 10 - ROC AUC modelo otimizado

Ao verificar a relevância de cada *feature* no modelo o resultado foi que 5 atributos acumulavam 70% da importância dos dados, como podemos ver na Figura 11.

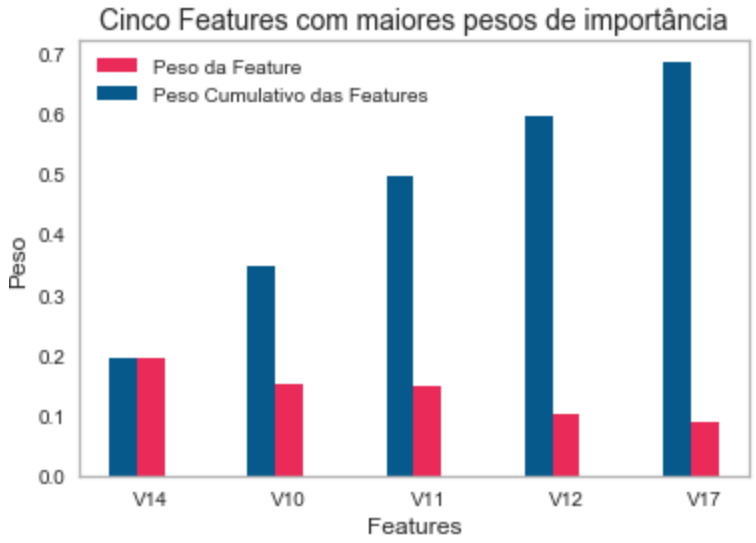

Figura 11 - Importância das Features

Contudo ao utilizar o mesmo modelo com apenas as 5 *features* o *ROC AUC* caiu para 0.9182, devido a grande diferença da métrica de ambos modelos, foi escolhido o modelo otimizado com todas as *features*.

Justificativa

Comparando o modelo final com benchmark e o modelo não otimizado, conclui-se que obteve uma melhora no resultado, sendo 95% no benchmark e 98% no modelo final, tais resultados podem ser vistos na Tabela 5.

	Benchmark	Modelo não otimizado	Modelo Otimizado
ROC AUC	95%	94%	98%

Tabela 5 - Comparação de modelos

V. Conclusão

Reflexão

Acredito que o projeto obteve sucesso na análise de detecção de fraudes por cartão de crédito, conseguindo melhorar um pouco o resultado demonstrado no benchmark. Foram utilizadas várias técnicas na qual necessitaram muita pesquisa para

seu entendimento e utilização. Outro ponto importante foi a necessidade de encontrar um modelo que tivesse um bom balanceamento na predição de ambas as classes, pois dos quatro algoritmos selecionados apenas o RF teve um bom desempenho (acima de 0,8) nas duas classes.

Para um próximo passo seria ideal tentar melhorar a predição do modelo para os Falsos Positivos, sem piorar o desempenho dos Verdadeiros Positivos, uma vez que neste estudo tivemos 72 casos de Falsos positivos, vistos na Figura 12.

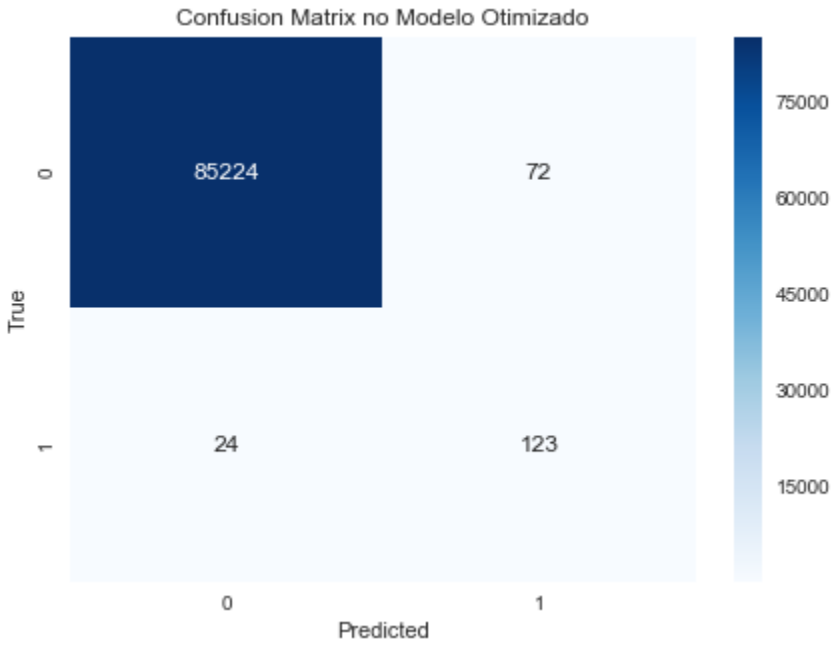

Figura 12 - Confusion Matrix

VI. Referências

Braga, Antonio Padua & Castro, Cristiano Leite de. 2011. **Aprendizado Supervisionado com conjunto de dados desbalanceados**

Castle, Nikki. 2017. **Supervised vs. Unsupervised Machine Learning**

Chen, Edwin. **Choosing a Machine Learning Classifier**

Donges, Niklas. 2018. **The Random Forest Algorithm**

Google Developers. 2018. **Classification: Precision and Recall**

Google Developers. 2018. **Classification: ROC and AUC**

Barella, Victor Hugo. 2015. **Técnicas para o problema de dados desbalanceados em classificação hierárquica**
Portella, Letícia.2018. **Machine Learning Models - My Cheat List**
Sachanm Lalit, 2015. **Logistic Regression vs Decision Trees vs SVM: Part II**
Statinfer, 2017. **SVM : Advantages Disadvantages and Applications**
Simplilean. 2018. **KNN Algorithm**
Singh, Aishwarya. 2018. **A Comprehensive Guide to Ensemble Learning**
Souza, Jocelyn D'. 2018. **Let's learn about AUC ROC Curve!**