DEEP LEARNING PARA DETECÇÃO DE FRAUDES EM CARTÕES DE CRÉDITO

DCA0121 - INTELIGÊNCIA ARTIFICIAL APLICADA

POR QUÊ ANALISAR CARTÕES DE CRÉDITO?

Compras com cartões de crédito e débito crescem 6% no 1º trimestre de 2017

Volume movimentado somou R$ 285 bilhões, segundo a Abecs. Uso dos carões está presente em 28,2% das famílias brasileiras.

Fonte: G1, 2017

10/07/2017 às 16h32

Cartões de crédito e débito movimentaram mais de R$ 1 trilhão em 2016

Fonte: valor, 2017

CARTÕES DE CRÉDITO: O LADO OBSCURO

Brasil é o segundo país com mais fraudes em cartões, diz pesquisa

Quase metade da população já teve algum problema nos últimos cinco anos

Fonte: oglobo, 2016

Fraudes em cartão de crédito já passam de 920 mil desde o início do ano

Fonte: uol, 2018

"Este é um mal inerente ao comércio eletrônico, e infelizmente não há e-commerces à prova de fraude."

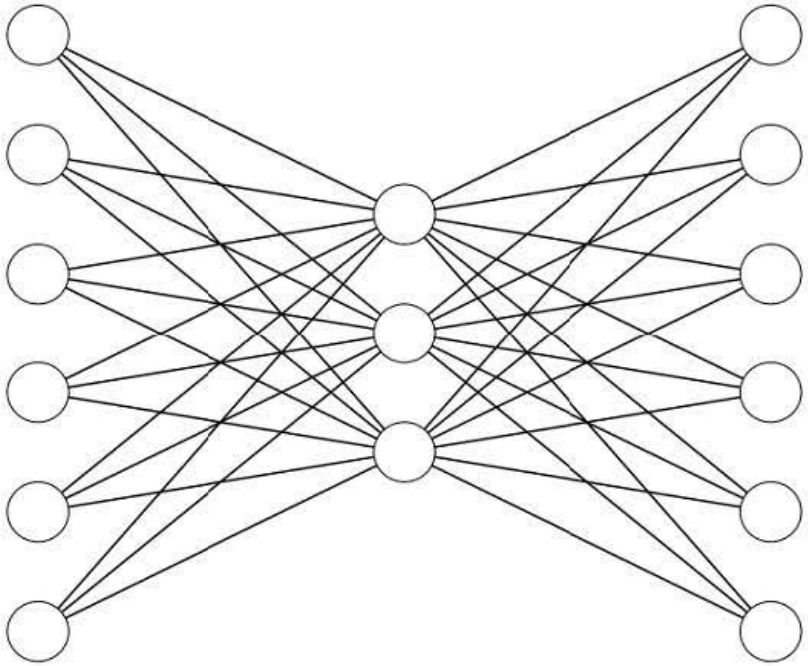
Fonte: e-commerce, 2018

POR QUE USAR MACHINE LEARNING?

- Os algoritmos mais tradicionais para detecção de fraudes estão sendo facilmente quebrados atualmente
- A indústria de serviços financeiras está confiando no aumento da complexidade desses algoritmos, e já estão utilizando alguns algoritmos de aprendizagem de máquina
- A técnica mais atual (e mais complexa) para detecção de fraudes envolve o uso de *autoencoders*, usado em *deep learning*

AUTOENCODER

- Recebe uma entrada, reduz a seus elementos principais e recria a entrada a partir disso, mantendo as características essenciais
- Treinados de forma não-supervisionada
- Assume-se que no uso de autoencoders irregularidades ou anomalias implicarão num grande erro de reconstrução detectável



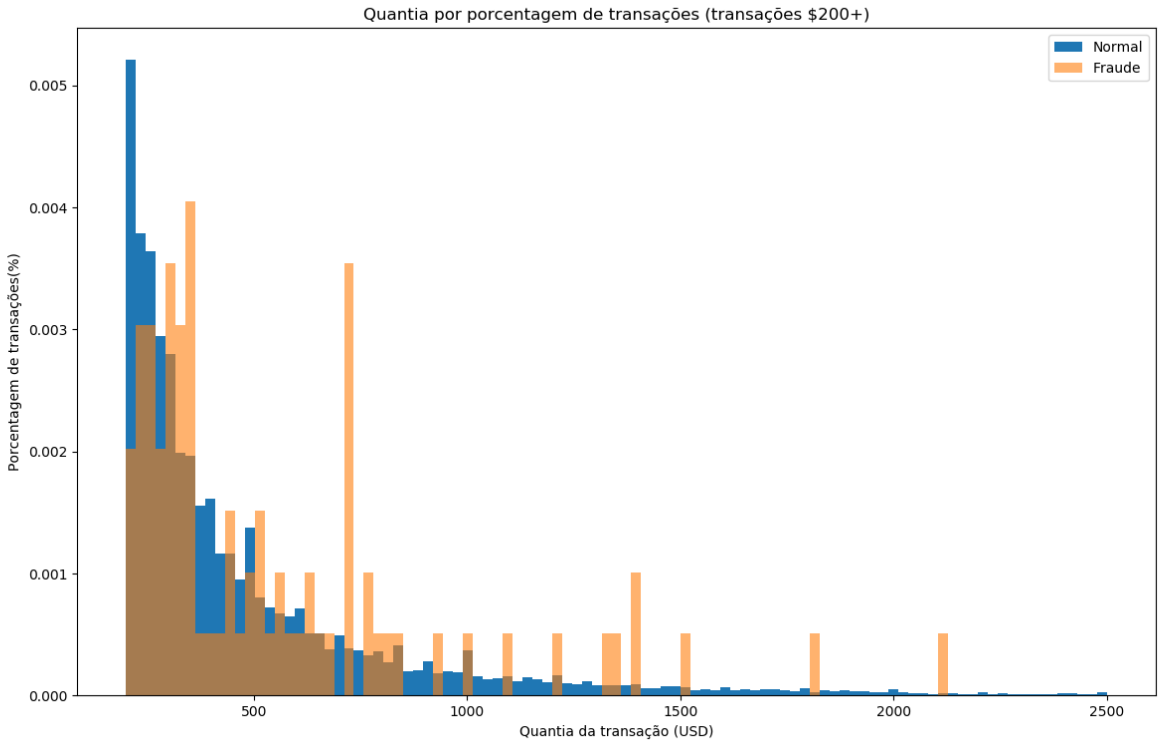
IMPLEMENTAÇÃO

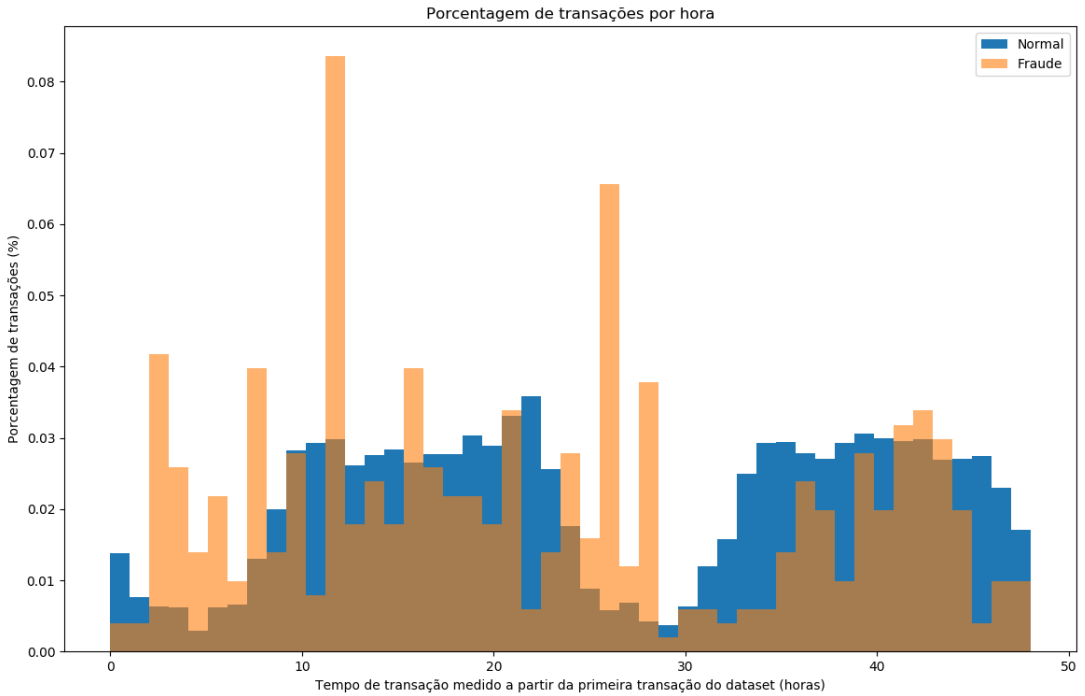
- Algoritmo em Python usando TensorFlow com Keras
- Dataset disponibilizado no Kaggle
- Dados de 284.807 transações com cartão de crédito realizadas num período de 48 horas, sendo 492 transações fraudulentas
- Cada transação inclui a quantia transacionada, o tempo desde a primeira transação e uma série de 28 parâmetros resultados de uma transformação PCA (devido a razões de confidencialidade)

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

IMPLEMENTAÇÃO

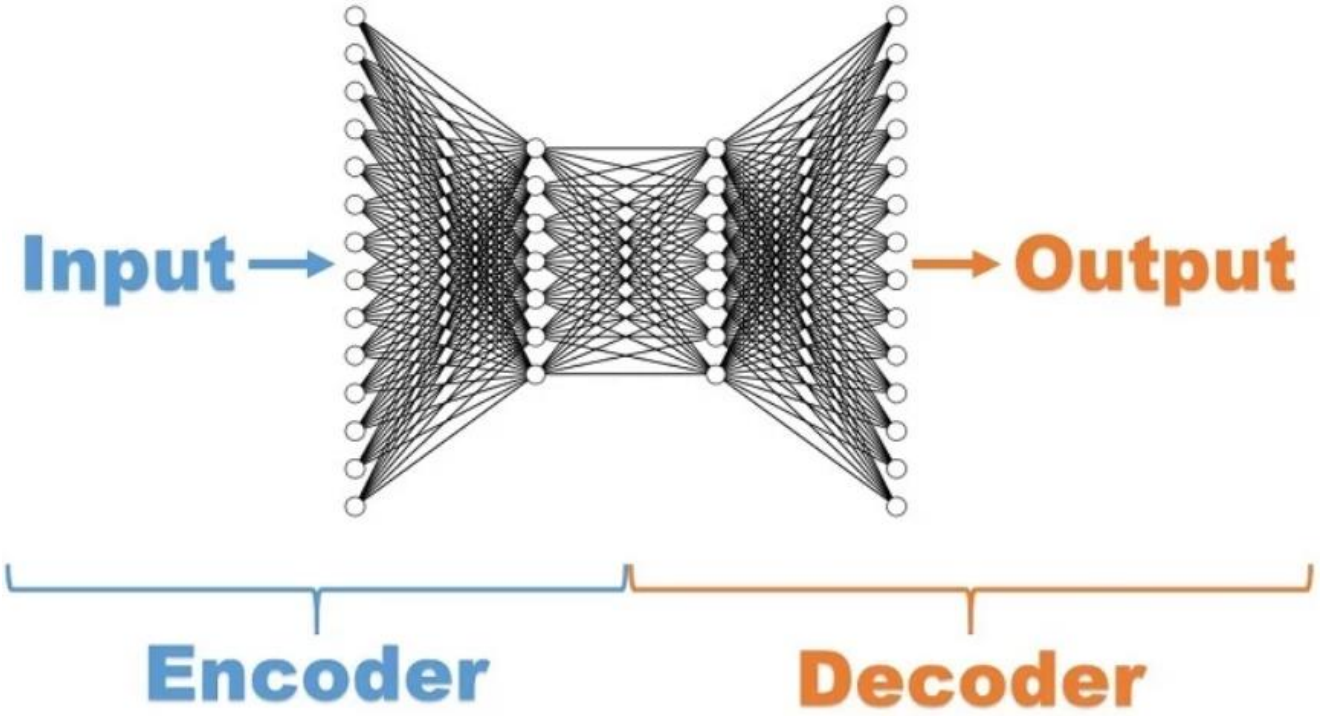
- A observação de gráficos dos tempos (horas) de transações e quantias transacionadas não são suficientes para se obter classificadores simples que indiquem transações fraudulentas





IMPLEMENTAÇÃO

- Criação do modelo da rede - encoder/decoder
- Parâmetros como taxa de aprendizado, funções de ativação, tamanho do batch e dimensão das camadas definidos empiricamente



IMPLEMENTAÇÃO

- Leitura do dataset
- Normalização e dimensionamento dos dados (diferentes magnitudes)
- Divisão do conjunto de treinamento e de teste (20% usado para teste)
- Criação do modelo e definição dos parâmetros da rede
- Treinamento do modelo
- Verificação dos resultados

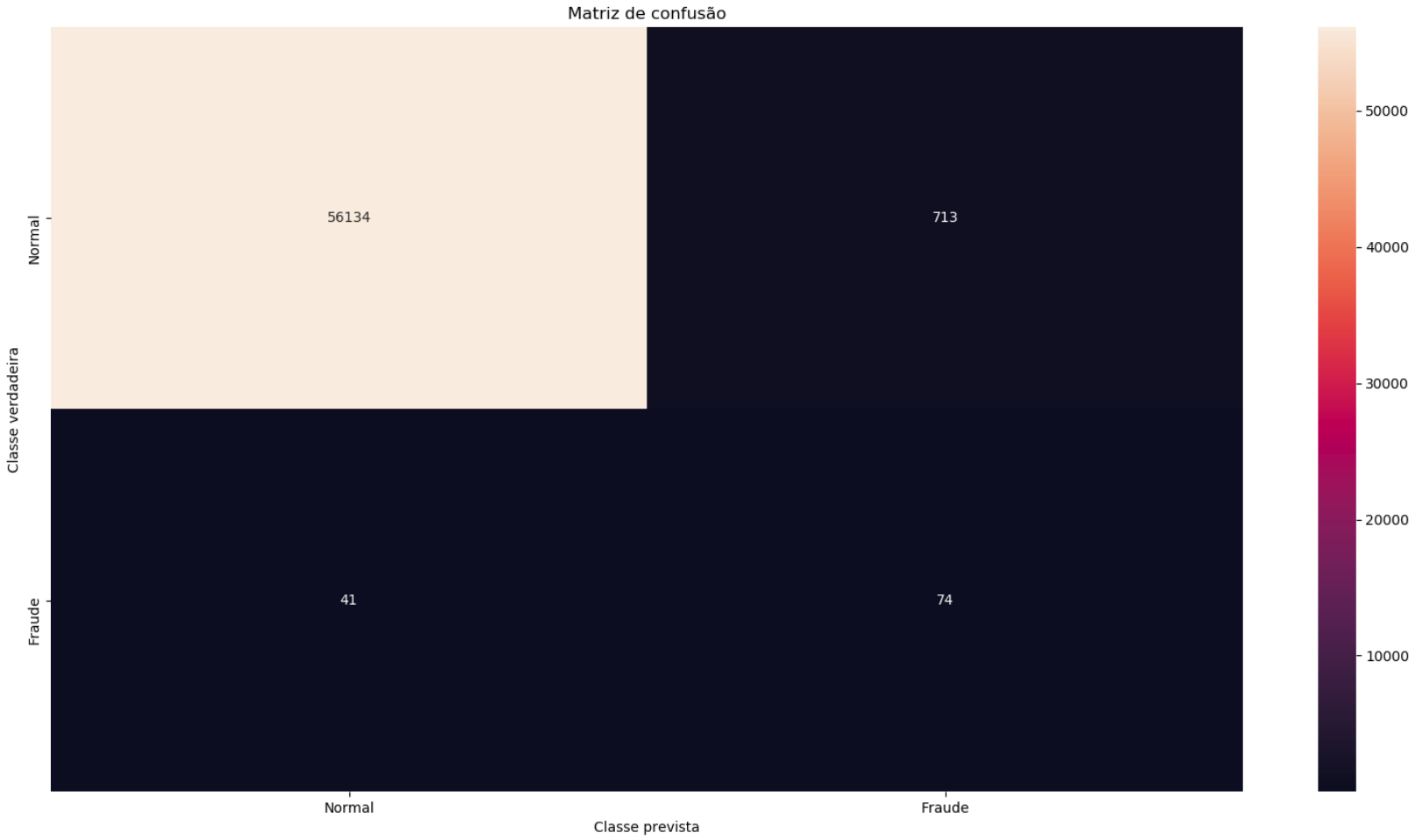
RESULTADOS

- A rede convergiu após cerca de 10 épocas
- Levou cerca de 8 segundos por época com aceleração GPU
- Chegou-se a detectar cerca de 60% dos casos de fraude

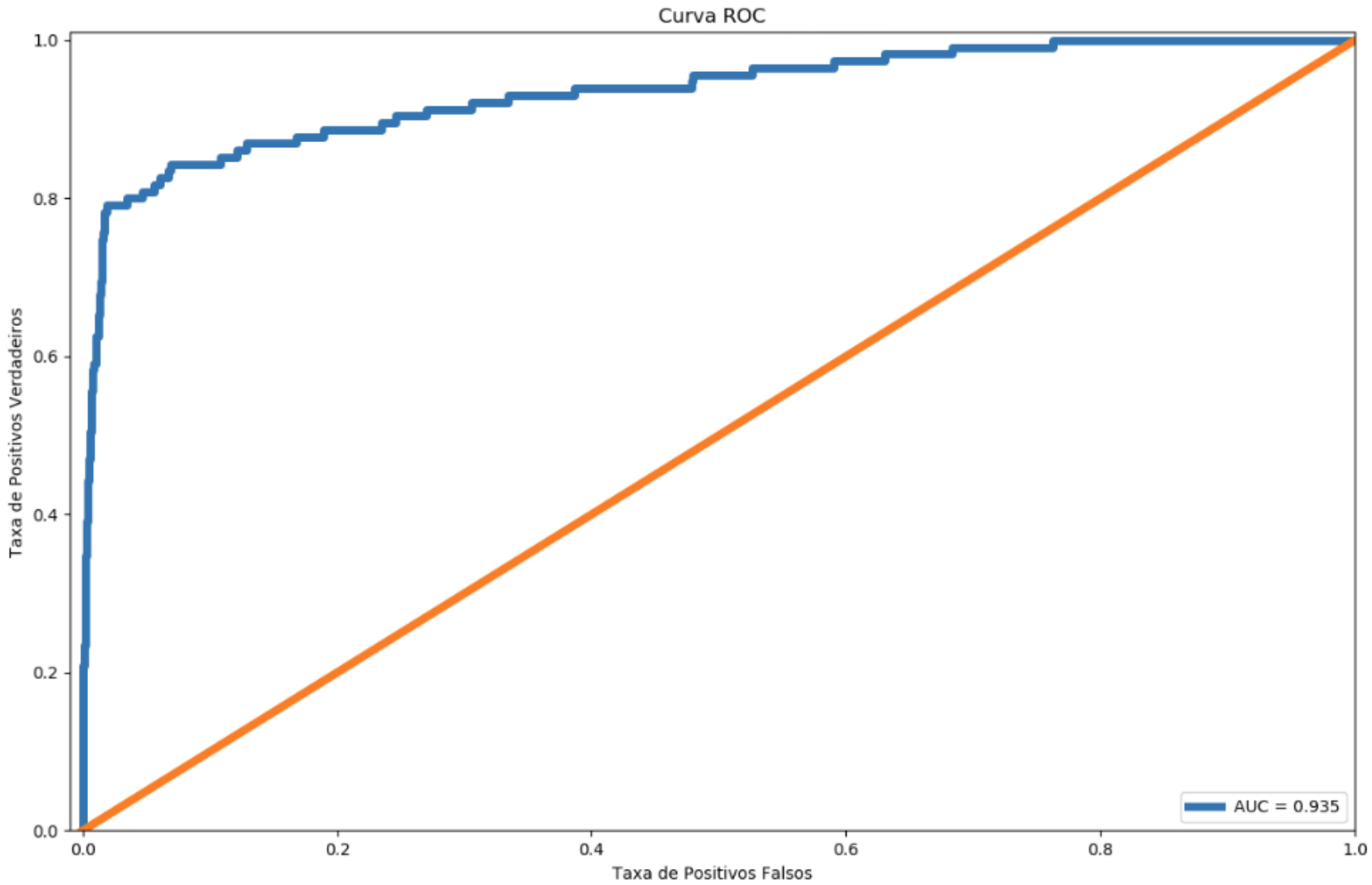
```
Epoch 1/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.8787 - acc: 0.4585 - val_loss: 0.8279 - val_acc: 0.5742
Epoch 2/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7746 - acc: 0.6062 - val_loss: 0.7952 - val_acc: 0.6285
Epoch 3/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7479 - acc: 0.6441 - val_loss: 0.7741 - val_acc: 0.6505
Epoch 4/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7326 - acc: 0.6575 - val_loss: 0.7640 - val_acc: 0.6609
Epoch 5/20
227468/227468 [==============================] - 8s 35us/step - loss: 0.7238 - acc: 0.6662 - val_loss: 0.7587 - val_acc: 0.6667
Epoch 6/20
227468/227468 [==============================] - 9s 37us/step - loss: 0.7183 - acc: 0.6729 - val_loss: 0.7516 - val_acc: 0.6720
Epoch 7/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7138 - acc: 0.6808 - val_loss: 0.7472 - val_acc: 0.6817
Epoch 8/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7104 - acc: 0.6881 - val_loss: 0.7434 - val_acc: 0.6911
Epoch 9/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7078 - acc: 0.6914 - val_loss: 0.7415 - val_acc: 0.6909
Epoch 10/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7059 - acc: 0.6950 - val_loss: 0.7395 - val_acc: 0.6944
Epoch 11/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7046 - acc: 0.6966 - val_loss: 0.7390 - val_acc: 0.6950
Epoch 12/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7035 - acc: 0.6977 - val_loss: 0.7376 - val_acc: 0.7009
Epoch 13/20
227468/227468 [==============================] - 8s 37us/step - loss: 0.7025 - acc: 0.6993 - val_loss: 0.7374 - val_acc: 0.6989
Epoch 14/20
227468/227468 [==============================] - 8s 36us/step - loss: 0.7017 - acc: 0.7019 - val_loss: 0.7361 - val_acc: 0.6963
Epoch 15/20
227468/227468 [==============================] - 8s 35us/step - loss: 0.7009 - acc: 0.7025 - val_loss: 0.7362 - val_acc: 0.7065
Epoch 16/20
227468/227468 [==============================] - 8s 35us/step - loss: 0.7004 - acc: 0.7037 - val_loss: 0.7352 - val_acc: 0.6990
Epoch 17/20
227468/227468 [==============================] - 8s 35us/step - loss: 0.7005 - acc: 0.7048 - val_loss: 0.7353 - val_acc: 0.7019
Epoch 18/20
227468/227468 [==============================] - 8s 34us/step - loss: 0.6997 - acc: 0.7053 - val_loss: 0.7352 - val_acc: 0.7023
Epoch 19/20
227468/227468 [==============================] - 8s 35us/step - loss: 0.6992 - acc: 0.7066 - val_loss: 0.7338 - val_acc: 0.7064
Epoch 20/20
227468/227468 [==============================] - 8s 35us/step - loss: 0.6991 - acc: 0.7070 - val_loss: 0.7342 - val_acc: 0.7047
```

RESULTADOS

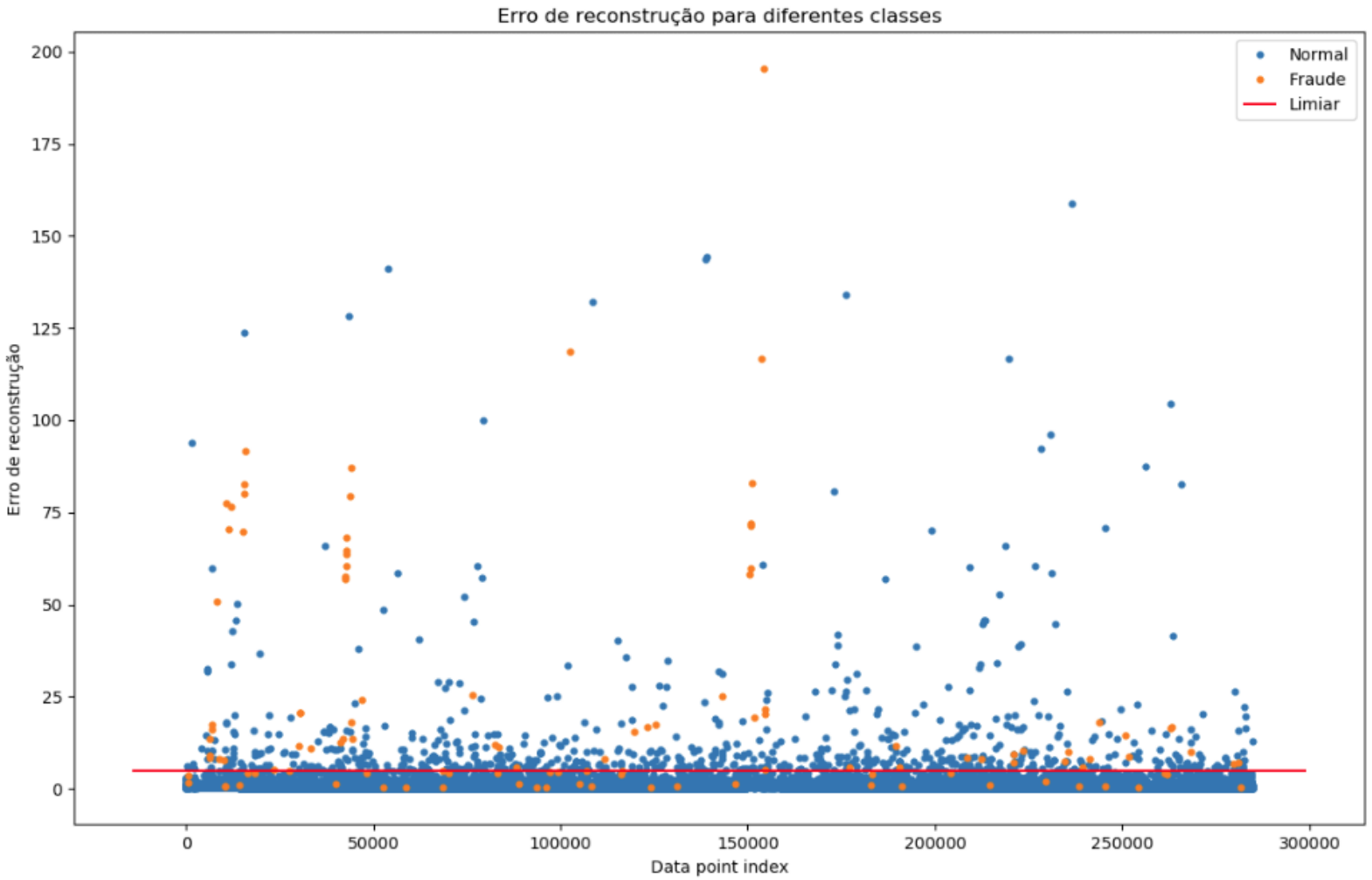
- O dataset desbalanceado pode ser um dos motivos para não se ter um resultado mais preciso
- Devido a isso algumas curvas de análise se tornam menos úteis
- Utilização de técnicas de sub sample para igualar os casos
- É uma ferramenta que deve ser utilizado como auxílio a outras de um conjunto de detectores de fralde



MATRIZ DE CONFUSÃO


Curva ROC
Taxa de Positivos Verdadeiros
Taxa de Positivos Falsos
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.935

CURVA ROC



ERRO DE RECONSTRUÇÃO

REFERÊNCIAS

- [1] https://economia.uol.com.br/noticias/estadao-conteudo/2018/09/25/fraudes-em-cartao-de-credito-ja-passam-de-920-mil-no-pais-desde-o-inicio-do-ano.htm?cmpid=copiaecola
- [2]https://economia.uol.com.br/noticias/estadao-conteudo/2018/09/25/fraudes-em-cartao-de-credito-ja-passam-de-920-mil-no-pais-desde-o-inicio-do-ano.htm?cmpid=copiaecola
- [3]https://www.valor.com.br/financas/5032900/cartoes-de-credito-e-debito-movimentaram-mais-de-r-1-trilhao-em-2016
- [4]https://g1.globo.com/economia/seu-dinheiro/noticia/compras-com-cartoes-de-credito-e-debito-crescem-6-no-1-trimestre-de-2017.ghtml
- [5] https://oglobo.globo.com/economia/brasil-o-segundo-pais-com-mais-fraudes-em-cartoes-diz-pesquisa-19752487
- [6]https://www.ecommercebrasil.com.br/noticias/e-commerce-brasileiro-sofre-uma-tentativa-de-fraude-cada-cinco-segundos/
- [7]https://www.datascience.com/blog/fraud-detection-with-tensorflow?fbclid=IwAR2vxrhKem6AuKyTj8xJoRaITXr-96F2HjKP0Em_cWBTn_wmDYL6RLNmskg