Engenheiro de Machine Learning Nanodegree

Projeto Final – Udacity **Danilo Ciciliotti, 04/08/2019**

COMPARAÇÃO ENTRE RANDOM FOREST E REDE NEURAL AUTOENCODER PARA DETECÇÃO DE FRAUDES FINANCEIRAS

I. DEFINIÇÃO

Visão Geral

A Associação CreditCards.com recentemente descobriu que a fraude de compras utilizando cartões de crédito está até 2020 exponencialmente em ascensão. Tanto o número de fraudes quanto os tipos de fraudes com cartões de crédito são cada vez maiores. Infelizmente, esses números parecem destinados a crescer ainda mais no futuro. Especialistas preveem que as fraudes com cartões de crédito ultrapassem 30 bilhões de dólares ainda em 2020 [1]. Por esse motivo, é importante encontrar uma maneira de reconhecer automaticamente as anomalias. Muitas pesquisas foram feitas para encontrar uma solução para esse problema. Referências [2] e [3] propõem soluções usando Redes Neurais Artificiais.

Figura 1: Fraudes de cartão de crédito em todo mundo (2010 – 2020) [1]

Enunciação do Problema

O objetivo do sistema de detecção de fraudes é detectar a fraude com precisão e antes que a fraude seja cometida. No que se refere à detecção de fraudes em cartões de crédito, o entrave de classificação requer a desenhar modelos com IA suficiente para filtrar adequadamente as transações como sendo legítima ou fraudulenta. As técnicas mais comuns usadas para detecção de fraudes são *Nave Bayes* (NB), *Support Vector Machines* (SVM) e *K-Nearest Neighbor Algorms* (KNN). Cada transação possui um conjunto de recursos exclusivos, como o valor da transação, a

hora em que ocorreu, o emissor e o destinatário, um id e outros dados confidenciais. O problema será estruturado como um problema de classificação, onde cada transação pode ser classificada como “Normal” ou “Fraude”. Com este projeto, será comparado as performances de dois modelos: *Autoencoders* e Florestas Aleatórias (*Random Forests*) na classificação de transações financeiras fraudulentas.

Métricas de avaliação

Precisão e Recall são uma métrica comum para esse tipo de problema; valores decentes de ambos ajudam a obter um sistema preciso que reconheça a maioria das transações ruins. Vamos definir VP, FP, VN e FN, respectivamente, verdadeiros positivos, falsos positivos, verdadeiros negativos e falso negativos.

$$Precisão = \frac{VP}{VP + FP}$$

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

Para resolver o problema da maneira correta, preferimos um alto valor de recall (o que significa que muitas fraudes foram detectadas). Também levamos em consideração a curva ROC e a matriz confusão, que nos ajudarão a avaliar os desempenhos. O objetivo de um sistema de detecção de fraude é identificar corretamente transações de fraudes com um número baixo de falsos negativos. Podemos até aceitar alguma transação normal que é reconhecida como fraude, mas precisamos classificar corretamente as transações que são realmente fraudes. A precisão e o recall nos ajudarão a avaliar o comportamento do sistema. A curva ROC em particular nos ajudará a fixar os valores corretos da memória de recuperação e a matriz de confusão será uma maneira útil de medir os resultados com precisão.

II. Análise

Exploração de Dados

Para reproduzir esse tipo de problema, encontrei um conjunto de dados útil disponível no Kaggle [4]. Os conjuntos de dados contêm transações efetuadas por cartões de crédito em setembro de 2013 por portadores de cartões europeus. Este conjunto de dados apresenta transações que ocorreram em dois dias, onde temos 492 fraudes de 284.807 transações. O conjunto de dados é altamente desequilibrado e a classe positiva (fraudes) representa 0,172% de todas as transações. Ele contém apenas variáveis numéricas de entrada que são o resultado de uma transformação do PCA. Devido a questões de confidencialidade, os autores não podem fornecer os recursos originais e mais informações básicas sobre os dados. Os recursos V1, V2, ... V28 são os principais componentes obtidos com o PCA, as únicas features que não foram transformados com o PCA são "tempo" e "quantidade". Features "tempo" contém os segundos decorridos entre cada transação e a primeira transação no conjunto de dados. A feature "valor" é o montante da transação. Feature "classe" é a variável resposta e leva valor 1 em caso de fraude e 0 caso contrário.

Visualização Exploratória

Temos um conjunto de dados altamente desequilibrado (Figura 1). De fato, o número de transações normais é exponencialmente maior que as transações fraudulentas.

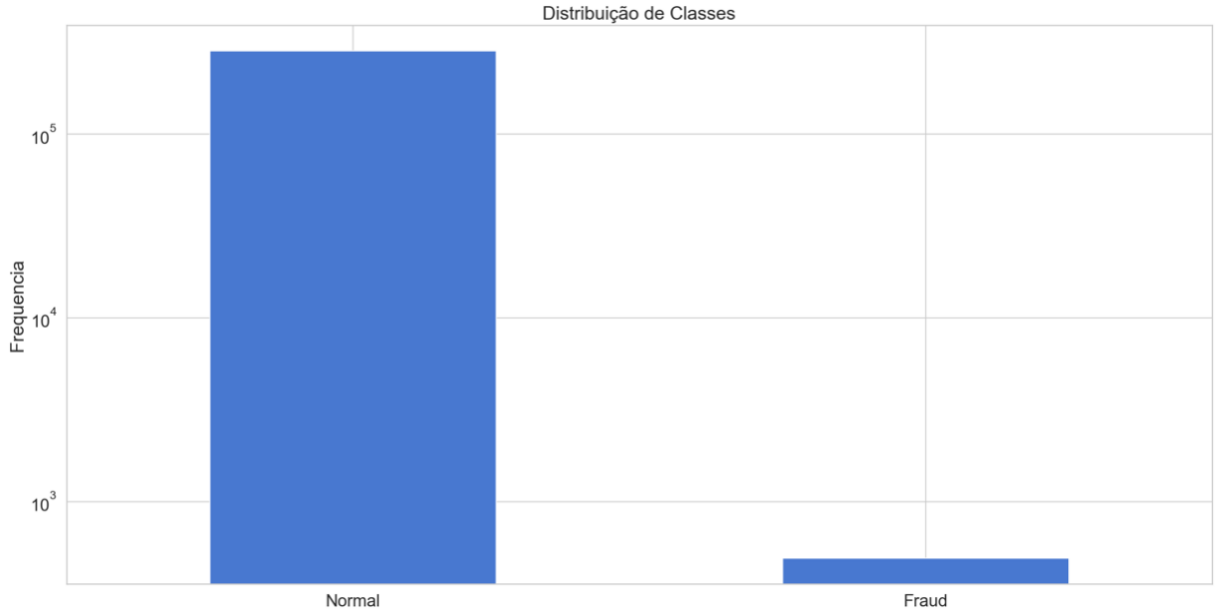

Figura 2: Distribuição de Classes

A quantidade por porcentagem de transações para cada classe é representada pela Figura 3. Podemos supor que esse recurso não é tão importante. A distribuição é quase a mesma para as duas classes.

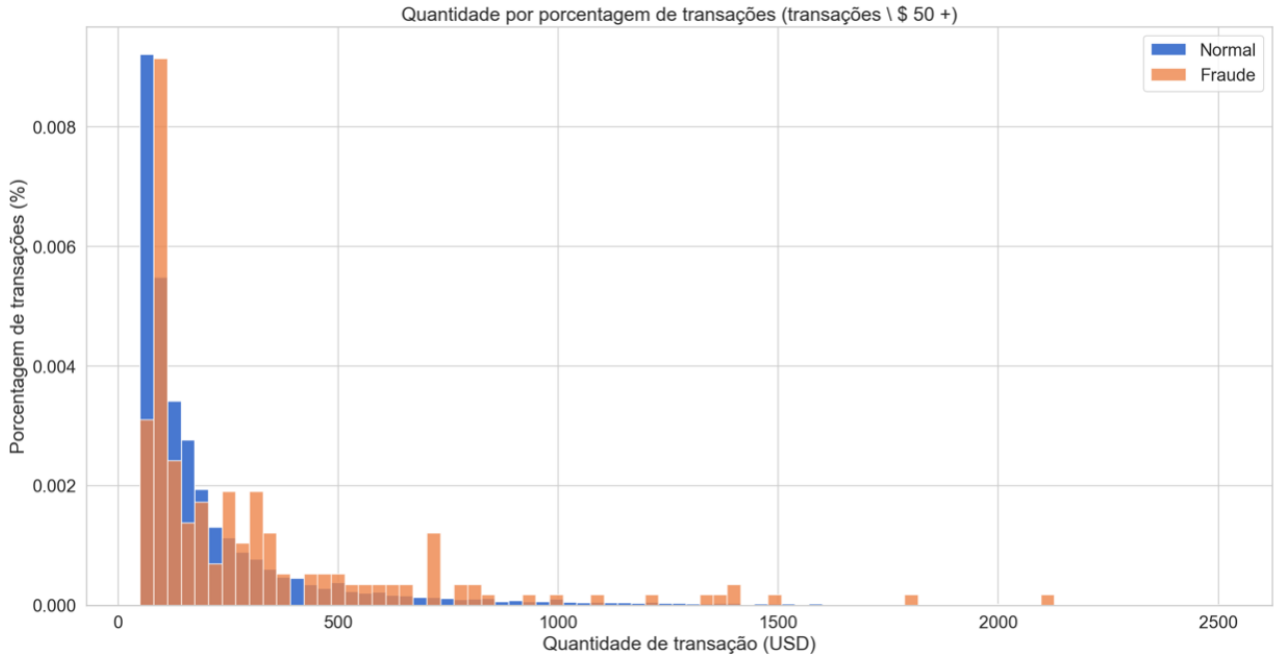

Figura 3: Quantidade por porcentagem de transações.

A fim de obter uma melhor compreensão do significado de cada característica, podemos analisar a distribuição de densidade de cada uma das classes. Os resultados da Figura 4 podem nos ajudar a selecionar os recursos certos e ignorar aqueles que não são relevantes.

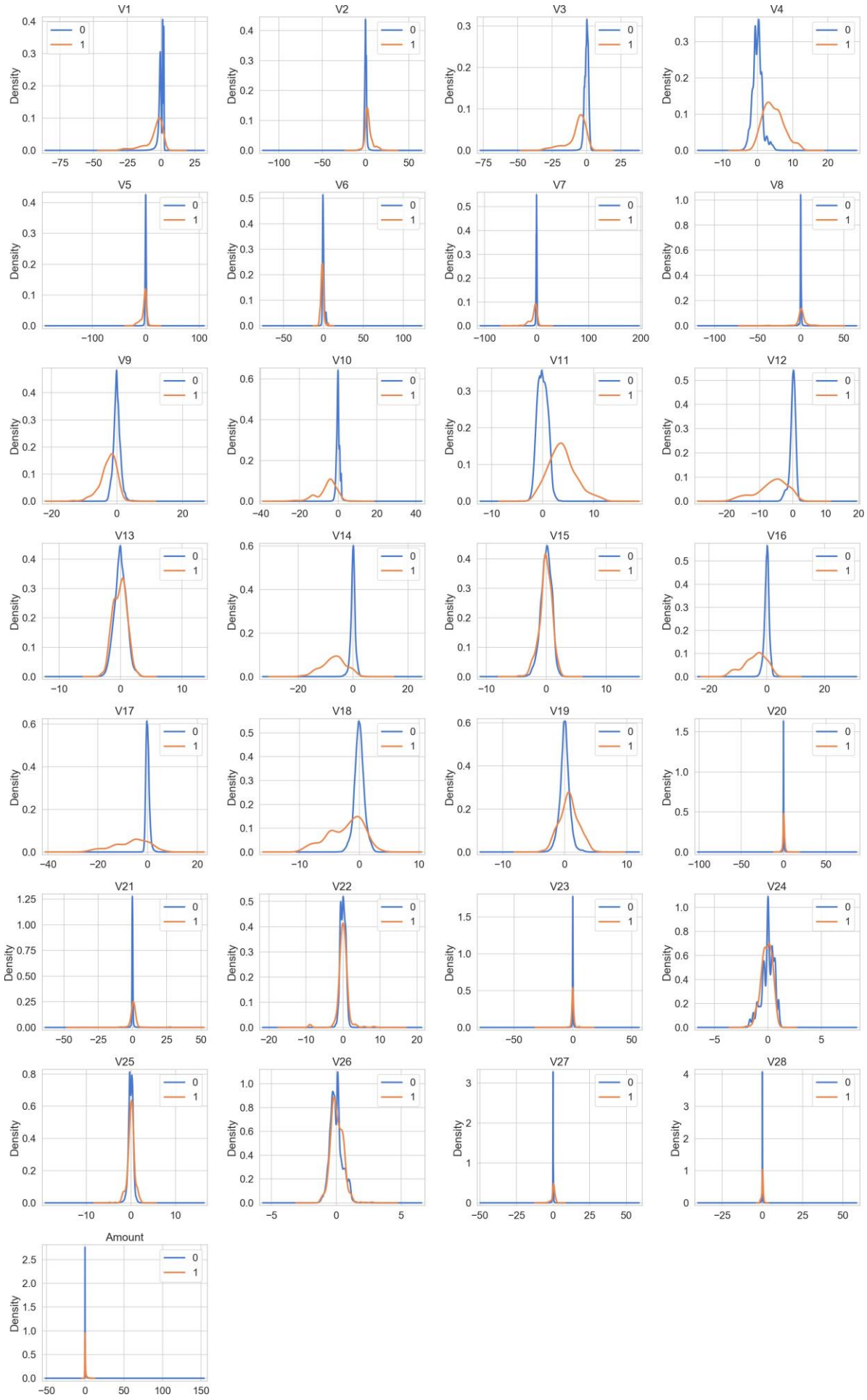

Figura 4: Distribuição de densidade de cada recurso

Técnicas e Algoritmos

Uma Rede Neural Artificial (RNA) é um modelo de processamento de informação que foi inspirado no modo como o cérebro processa as informações. Cada elemento de entrada é tratado como um grande número de elementos de processamento altamente interconectados (neurônios) trabalhando juntos para resolver problemas específicos. Aprender em sistemas biológicos envolve ajustes nas conexões sinápticas que existem entre os neurônios. Isso também é verdade para as RNAs. A rede aprende os pesos de cada conexão de forma que o erro de saída seja minimizado. Uma rede neural geralmente é composta por uma camada de entrada, um número variável de camadas ocultas e a camada de saída. Você pode representar uma rede neural como uma função de funções. Cada camada representa uma função específica, cuja entrada é fornecida pela camada anterior e a saída é entregue à próxima camada. A informação flui apenas da entrada para a saída.

Os autoencoders são uma classe particular de redes neurais que captam uma entrada, "comprimem" essa entrada até os recursos principais e tentam reconstruir a entrada original a partir dessa representação comprimida. A entrada é geralmente mapeada em um espaço menor ou maior a partir da primeira metade da rede. Essa primeira fase é chamada de codificação e produz uma representação modificada da entrada original. A segunda parte da rede "decodifica" essa representação e tenta reconstruir a entrada original na camada de saída. Para configurar um autoencoder, precisamos selecionar um determinado número de camadas e funções de ativação, a taxa de aprendizado na qual a rede opera e a dimensão de codificação de cada camada. Este modelo é usado para aplicações em que queremos reconhecer anomalias ou “distorções” nos dados. Podemos usar o autoencoder da seguinte maneira: nós o treinamos para aprender a reconhecer transações normais. Durante a primeira fase, a rede codifica a entrada original (ou seja, uma transação) em um espaço maior; depois, na fase de decodificação, tenta reconhecer a representação e reconstruir a entrada original.

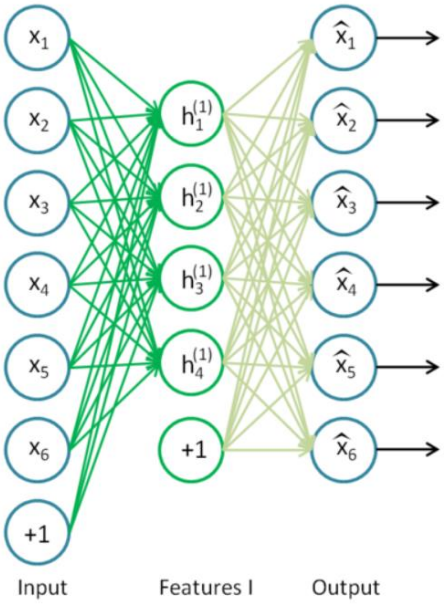

Figura 5: A estrutura de um autoencoder [4]

O aprendizado de uma árvore de decisão (*decision tree*) realiza observações sobre um item (representado como ramificações) até conclusões sobre o valor de destino do item (folhas). As árvores que crescem muito profundamente tendem a aprender padrões altamente irregulares:

elas superam seus conjuntos de treinamento, ou seja, têm um viés baixo, mas uma variação muito alta. As florestas aleatórias (*random forests*) é um método de aprendizado conjunto para a classificação que opera através da construção de uma multiplicidade de árvores de decisão no treinamento e treinado em diferentes partes do mesmo conjunto de treinamento, com o objetivo de reduzir a variância. Dessa forma, as florestas de decisão aleatórias corrigem o hábito das árvores de decisão de sobrecarregar seu conjunto de treinamento. Você pode configurar um calibrador de floresta aleatório ajustando o número de estimadores, a profundidade máxima, o número mínimo de amostras necessárias para dividir um nó interno ou estar em um nó em folha e um peso de classe específico. Esse tipo de modelo combina perfeitamente com problemas de classificação como o que estamos tentando resolver. Assim, ele será usado para aprender a classificar transações como normal ou fraude.

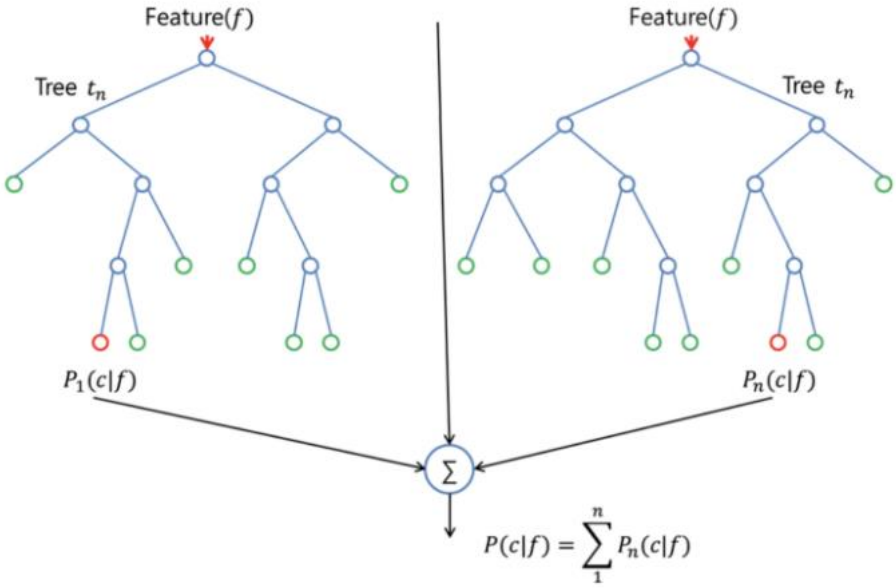

Figura 6: A estrutura de uma Floresta Aleatória (Randon Forests) com duas árvores [5]

O principal desafio quando se trata de modelar a detecção de fraudes financeiras como um problema de classificação vem do fato de que, nos dados do mundo real, a maioria das transações não é fraudulenta. Isso nos traz um problema: dados desbalanceados. Há vários métodos disponíveis para fazer tratamento do conjunto de dados. A técnica mais comum é conhecida como SMOTE*: Synthetic Minority Over-sampling Technique* (Técnica de Super-Amostragem Sintética da Minoria). Essa técnica será usada para balancear o conjunto de dados como uma etapa de pré-processamento.

Modelo Benchmark

Para medir a qualidade deste experimento e avaliar os resultados, precisamos de algum modelo de referência. Parece realmente difícil encontrar outros que conduziram e compartilharam o resultado desse tipo de análise. Após algumas pesquisas, encontrei duas outras soluções exatamente no mesmo conjunto de dados que eu selecionei. A referência [6] reconhece as fraudes usando a regressão logística com 93,5% de precisão, enquanto a referência [7] usa os classificadores KNN e Naive Bayes com precisão de 99,8% e 97,7% respectivamente. A melhor maneira de avaliar este projeto será comparar a curva ROC que eu obtenho com as que estão abaixo na Figura 2.

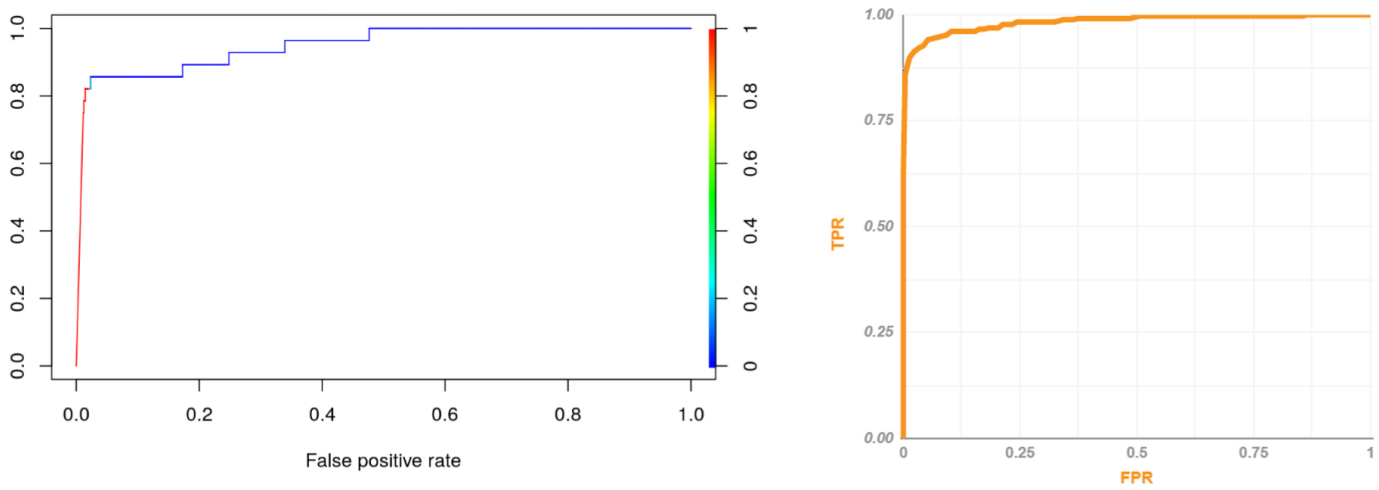

Figura 7: Curvas ROC de Modelos de Benchmark. [6] e [7]

Usarei esses resultados como modelos de referência. Eu não espero superar os resultados obtidos com o KNN ou o Naive Bayes, mas acho que os autoencoder alcançarão pelo menos os mesmos resultados do modelo [6].

	Normal	Fraude
Normal	99,64%	0,19%
Fraude	0,02%	0,15%

Tabela 1: Matriz Confusão para referência [6]

Para comparar numericamente os resultados, também levamos em consideração as matrizes de confusão das referências [5] e [6]. Observe que os valores foram normalizados como porcentagens devido ao tamanho diferente dos conjuntos de testes nos quais os resultados foram calculados.

	Normal	Fraude
Normal	97,57%	0,04%
Fraude	2,23%	0,16%

Tabela 2: Matriz Confusão para referência [7]

III. Metodologia

Pré-processamento de Dados

Observe que, no conjunto de dados, "Tempo" são os segundos decorridos entre cada transação e a primeira transação no conjunto de dados. É desconhecido durante a hora do dia em que as transações realmente começaram. Então, a coluna pode, no máximo, nos informar o quanto as transações foram fechadas entre duas fraudulentas. Desde que eu quero prever fraudes, independentemente do tempo de transação, esse recurso não é tão representativo, portanto, será removido. A maioria dos dados resulta do produto de uma análise de PCA. Devido a questões de confidencialidade, os autores não podem fornecer os recursos originais e mais informações básicas sobre os dados. Recursos V1, V2, V3, ..., V28 são os principais componentes obtidos com o PCA. Observe que “Quantidade” é o único recurso que não foi modificado. Por isso, normalizamos esse. Agora podemos avaliar a importância de cada recurso da Figura 4. A partir desta exploração visual, podemos remover os seguintes recursos: ”V13”, “V15”, “V22”, “V23”, “V26” e “Quantidade”.

Implementação

Os classificadores foram treinados nos dados de treinamento pré-processados. Isso foi feito em um Jupyter Notebook usando Python 3. As seguintes bibliotecas foram usadas:

- keras: para implementar o autoencoder;
- sklearn: implementar a floresta e métricas aleatórias
- imblearn: para implementar o SMOTE oversampling;
- mathplotlib: para plotar os gráficos;
- pandas e numpy: para processar facilmente os dados.

Primeiro, os dados foram carregados e divididos em conjuntos de treinamento e teste (20% do conjunto de dados). O treinamento ainda foi dividido a fim de usar os 10% para validação. O autoencoder foi desenvolvido usando keras. Foi necessário um pouco de esforço para projetar uma rede que aprende sem sobrecarregar (*overfitting*). Na Figura 8 podemos conferir a estrutura da rede. Tem quatro camadas ocultas. Durante a fase de codificação, mapeamos a entrada para 28 e 56 neurônios, depois tentamos reconstruir a entrada original com a fase de decodificação (ou seja, duas camadas ocultas de 56 e 28 neurônios, respectivamente, e 23 neurônios na saída).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 23)                0
_________________________________________________________________
dense_11 (Dense)             (None, 28)                672
_________________________________________________________________
dense_12 (Dense)             (None, 56)                1624
_________________________________________________________________
dense_13 (Dense)             (None, 56)                3192
_________________________________________________________________
dense_14 (Dense)             (None, 28)                1596
_________________________________________________________________
dense_15 (Dense)             (None, 23)                667
=================================================================
Total params: 7,751
Trainable params: 7,751
Non-trainable params: 0
_________________________________________________________________
```

Figura 8: O autoencoder, feito de 4 camadas ocultas e a saída.

O modelo foi treinado em conjuntos de treinamento e validação para 100 ciclos, definindo o tamanho do lote para 128 e a taxa de aprendizado para 0,001. A função de perda foi plotada usando o mathplotlib e o sistema foi avaliado no conjunto de testes usando as métricas acima. Ao final foi plotado uma matriz confusão para o modelo autoencoder.
Como mencionamos acima, queremos comparar os autoencoder com a Florestas Aleatórias (*Random Forests*). A fim de aumentar o desempenho deste modelo, também aplicamos aumento de dados. Dividimos os dados em conjuntos de treinamento, validação e teste, conforme descrito acima. Assim, aumentamos o conjunto de treinamento usando a função SMOTE do imblearn. A floresta aleatória foi treinada e implementada usando a biblioteca padrão sklearn. Ainda todas as

métricas de avaliação foram usadas como explicado anteriormente. Observe que o aumento de dados aumenta muito os desempenhos. Na verdade, experimentei também algumas outras técnicas, como subamostragem (*undersampling*), mas elas não forneceram os mesmos resultados.

Tratamento

A implementação exigiu muito tratamento. Comecei com uma rede simples, tentando comprimir a entrada original e extrair as informações mais relevantes. Eu configurei um autoencoder com a camada de codificação com 14 e 7 neurônios. Em seguida, configurei o tamanho do lote de 32 (por padrão) para 128. Inicialmente, usei todo o conjunto de recursos, exceto "Tempo". Quando removi recursos menos importantes, o nível de acurácia da rede aumentou.

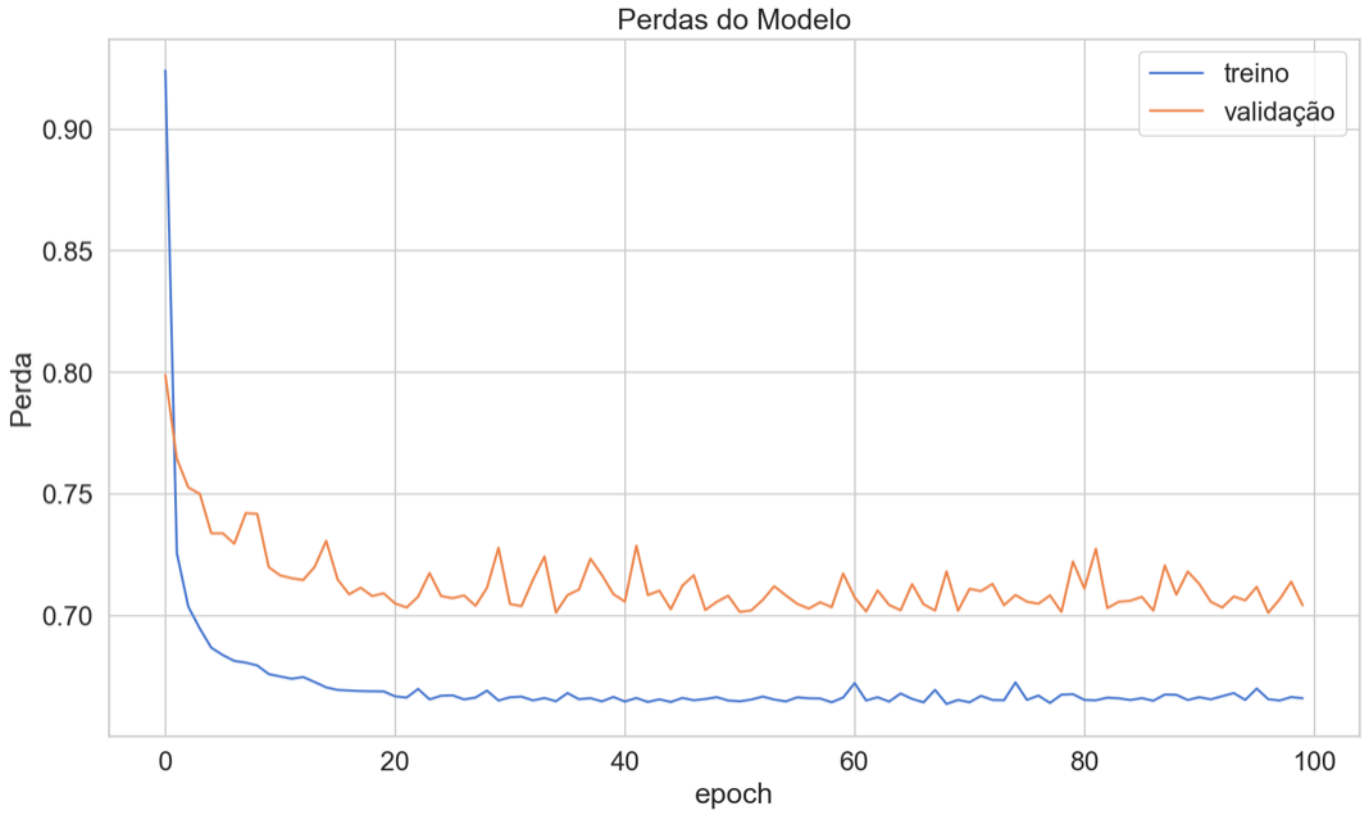

Figura 9: Função de perda do autoencoder.

Eu também tentei aplicar o Random Forests sem aumento de dados. Obviamente, os resultados não foram tão bons. Eu também experimentei técnicas de subamostragem, mas isso não ajudou.

IV. Resultados

Avaliação e Validação dos Modelos

Os resultados obtidos com os dois modelos são bastante interessantes. O autoencoder é menos preciso, mas reconhece corretamente uma boa porcentagem de fraudes. O modelo Random Forests, por outro lado alcança resultados ainda mais altos e com uma precisão melhor. Ambos

os modelos foram treinados e validados em mesmo subconjunto do conjunto de dados, enquanto os 20% do conjunto de dados foram separados para testes. A avaliação do modelo foi feita neste subconjunto, a fim de garantir a robustez dos modelos. A Figura 7 mostra as curvas de recall de precisão para ambos os modelos.

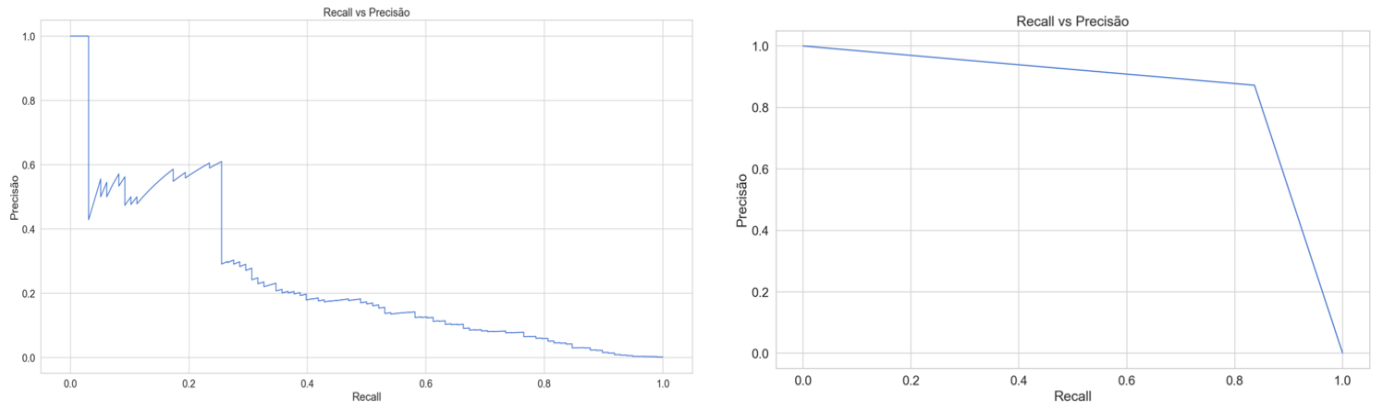

Figura 10: Curvas de Recall-Precisão dos modelos autoconder (esquerda) e random forests (direita).

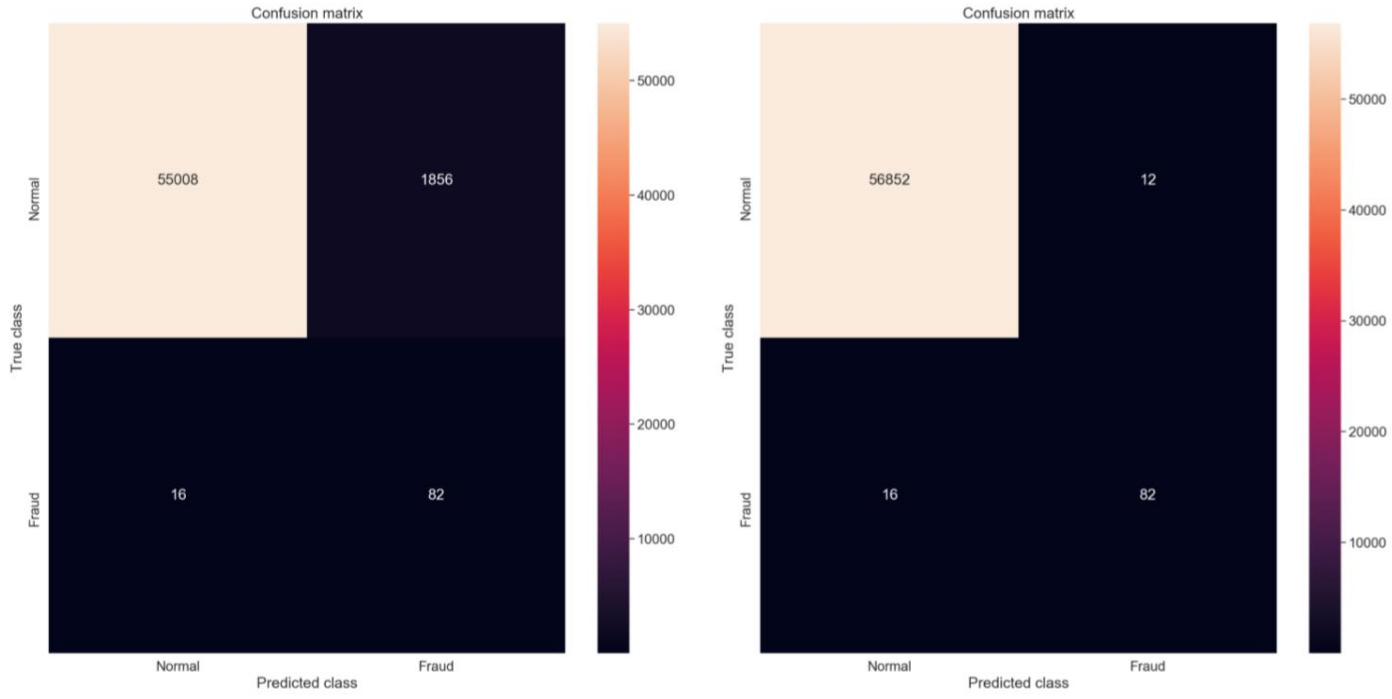

Figura 11: Matriz Confusão dos modelos autoconder (esquerda) e random forests (direita).

Para validar os dois modelos e validar sua robustez, podemos realizar uma validação cruzada k-fold. Como você pode ver na Tabela 3, a floresta aleatória se adapta perfeitamente a novas entradas, enquanto o autoencoder tem um forte desvio padrão, o que significa que o modelo não é tão confiável com dados novos e não vistos.

	Autoenconder	Random Forests
Média Acc.	86,63%	99,88%
Desvio	(+/- 6,87%)	(+/- 0,00%)

Tabela 3: Média e Desvio Padrão de ambos modelos

Justificativa

Para comparar os resultados com os valores de referência, precisamos normalizar as matrizes de confusão com porcentagens.

	Normal	Fraude
Normal	96,62%	3,26%
Fraude	0,03%	0,14%

Tabela 3: Matriz Confusão do autoencoder [7]

Apesar do autoencoder ter tido um bom resultado, ele foi completamente superado pelos dois modelos. Já o Random Forests é mais preciso que o modelo [7]. Ambos [6] e [7] têm um número maior de fraudes reais que foram detectadas.

V. Conclusão

Free-Form Visualization

Dos resultados acima podemos concluir que ambos os modelos aprenderam como reconhecer as fraudes. A Figura 11 ajuda-nos a ter uma ideia do equilíbrio da taxa de verdadeiros positivos / negativos do autoencoder e do random forests.

Reflexão

A detecção de fraudes é uma tarefa importante e sensível. Requer técnicas que reconhecem automaticamente anomalias com baixas taxas de erro. Na verdade, esses sistemas precisam identificar corretamente todas as transações ruins, possivelmente com poucos falsos positivos. A abordagem que propusemos neste projeto resolve corretamente o problema, mesmo que ele ainda deva ser melhorado. Como esperado, os autoenconder não são o melhor modelo para resolver esse tipo de problema. A rede obviamente aprende, mas não é tão precisa quanto outros modelos. Em vez disso, as Florestas Aleatórias (Randon Forests) são uma opção melhor para problemas de detecção de anomalias, mas é importante equilibrar o modelo para obter resultados decentes, portanto, requer um pouco de etapas de pré-processamento.

Melhoria

O objetivo deste projeto foi comparar um modelo “padrão” como florestas aleatórias com uma abordagem alternativa (autoenconder). Eu não esperava que o autoencoder superasse as técnicas mais estabelecidas. Outras melhorias podem ser obtidas usando técnicas alternativas. O modelo Random Forest pode ser melhorado com alguma outra experimentação. Espero ver alguma melhoria com a abordagem combinada para equilibrar o conjunto de dados usando métodos de classe combinadas SMOTE + ENN.

Referencias

[1] Report Nilson . Acessível em: https://nilsonreport.com/upload/content_promo/The_Nilson_Report_10-17-2016.pdf?source=post_page----------

[2] Credit Card Fraud Detection via KNN and Naive Bayes classifiers. Acessível em: https://www.kaggle.com/yuridias/credit-card-fraud-detection-knn-naive-bayes

[3] Redes neurais, o que são e suas importâncias. Acessível em: https://www.sas.com/pt_br/insights/analytics/neural-networks.html

[4] Redes Neurais de Terceira Geração: Redes Profundas. Acessível em: https://www.mql5.com/pt/articles/1103

[5] Aprendendo em uma Floresta Aleatória. Acessível em: https://medium.com/machina-sapiens/o-algoritmo-da-floresta-aleat%C3%B3ria-3545f6babdf8

[7] Credit Card Fraud Detection: Estudo de caso, por Ayushi Agrawal, Shiv Kumar e Amit Kumar Mishra. Acessível em:

https://www.kaggle.com/yuridias/credit-card-fraud-detection-knn-naive-bayes

[8] Estudo de caso; How to Implement Credit Card Fraud Detection Using Java and Apache Spark. Acessível em: https://www.romexsoft.com/blog/implement-credit-card-fraud-detection/