CODERHOUSE

Alan Ribeiro
Francisco Santos
Henrique Leitão
Leonardo Dressani

DETECÇÃO DE FRAUDES E TRANSAÇÕES DE CARTÃO DE CRÉDITO

SÃO PAULO
2023

Alan Ribeiro
Francisco Santos
Henrique Leitão
Leonardo Dressani

DETECÇÃO DE FRAUDES E TRANSAÇÕES DE CARTÃO DE CRÉDITO

Projeto de pesquisa apresentado ao curso de Data Science, da instituição de ensino Coderhouse, a ser utilizado como diretrizes para manufatura do Trabalho de Conclusão de Curso.

SÃO PAULO
2023

RESUMO

O objetivo desse trabalho é o desenvolvimento de uma análise de dados exploratória e modelos de machine learning capazes de prever possíveis fraudes em transações de cartão de crédito. Para isso foram utilizadas duas bases de dados, obtidas no site da Kaggle, sendo uma para treinamento e outra para teste e validação dos modelos. No referencial teórico foram desenvolvidos modelos de classificação em linguagem Python de programação e comparados um ao outro a fim de escolher o modelo mais adequado ao estudo. Como conclusão verificou-se diferentes resultados entre os modelos escolhidos e um melhor desempenho em detecção de fraudes do modelo LightGBM.

Palavras-chave: análise de dados exploratória, machine learning, Python, transações de cartão de crédito, detecção de fraudes, modelo LightGBM.

ABSTRACT

This study aims to develop an exploratory data analysis and machine learning models capable of predicting potential fraud in credit card transactions. Two datasets, acquired from Kaggle platform, were utilized for model training and testing/validation. In theoretical framework, classification models were implemented using the Python programming language and compared to identify the most suitable model for the study. The results revealed variations in performance among the chosen models, with the LightGBM model exhibiting superior predictive capabilities for fraud detection.

Keywords: exploratory data analysis, machine learning, Python, credit card transactions, fraud detection, LightGBM model.

Sumário

INTRODUÇÃO

A crescente digitalização das transações financeiras proporcionou uma comodidade sem precedentes, mas também intensificou os desafios associados à segurança, especialmente no cenário das transações de cartão de crédito. A detecção de fraudes tornou-se uma área crucial no âmbito da ciência de dados, onde o uso de técnicas avançadas tem se destacado como um meio eficaz de proteger os consumidores e as instituições financeiras.

No contexto das transações de cartão de crédito, a detecção de fraudes é um campo multidisciplinar que envolve a análise extensiva de dados para identificar padrões suspeitos e atividades não autorizadas. Este trabalho visa explorar e desenvolver métodos de detecção de fraudes por meio de técnicas de ciência de dados, com ênfase na análise exploratória de dados e na aplicação de modelos de machine learning.

O presente estudo se fundamenta na importância de mitigar os riscos associados às fraudes em transações de cartão de crédito, considerando o impacto financeiro e reputacional para os consumidores e as instituições. Utilizando duas bases de dados obtidas no site da Kaggle, uma destinada ao treinamento e outra para teste e validação dos modelos, buscamos aprimorar a eficácia na identificação de atividades fraudulentas.

No decorrer desta pesquisa, serão desenvolvidos e comparados modelos de classificação em linguagem Python, avaliando sua capacidade de predição e identificação de padrões anômalos. A escolha do modelo mais adequado será embasada em uma análise criteriosa, considerando métricas de desempenho e eficácia na detecção de fraudes.

Ao concluir este estudo, esperamos não apenas contribuir para o avanço na área de detecção de fraudes em transações de cartão de crédito, mas também fornecer insights valiosos para aprimorar a segurança e a confiabilidade das transações financeiras em um cenário cada vez mais digitalizado e interconectado.

1. MODELOS DE CLASSIFICAÇÃO

Modelos de classificação são algoritmos de aprendizado de máquina projetados para categorizar ou classificar instâncias de dados em diferentes classes ou categorias. Esses modelos são treinados com base em conjuntos de dados rotulados, nos quais as classes das instâncias são conhecidas. O objetivo é aprender padrões e relações nos dados para que, posteriormente, o modelo possa atribuir automaticamente rótulos ou classes a novas instâncias não rotuladas.

Em contextos como detecção de fraudes em transações de cartão de crédito, os modelos de classificação analisam diversas características das transações e aprendem a distinguir entre transações legítimas e fraudulentas. A escolha do modelo adequado é crucial para garantir uma classificação precisa e eficiente.

Neste estudo, serão explorados e comparados diversos modelos de classificação, incluindo GradientBoostingClassifier, KNeighborsClassifier, LGBMClassifier e RandomForestClassifier cada um com suas características específicas e abordagens para lidar com o desafio da detecção de fraudes em transações financeiras. Esses modelos desempenham um papel fundamental na automatização desse processo, contribuindo para a segurança e integridade das operações financeiras em um ambiente cada vez mais digital.

1.1. **GradientBoostingClassifier**

O `GradientBoostingClassifier` é um algoritmo de aprendizado de máquina que faz parte da família de métodos de boosting e é utilizado para tarefas de classificação. Esse algoritmo pertence à biblioteca scikit-learn, que é uma biblioteca popular de aprendizado de máquina em Python.

Aqui estão alguns pontos-chave sobre o `GradientBoostingClassifier`:

1.1.1. Boosting:

O `GradientBoostingClassifier` é um modelo de boosting. Boosting é uma técnica de ensemble learning em que vários modelos fracos (geralmente árvores de decisão rasas) são combinados para formar um modelo mais robusto e preciso.

1.1.2. Gradient Boosting:

O termo "Gradient" em "GradientBoostingClassifier" refere-se ao fato de que esse algoritmo otimiza a função de perda do modelo usando gradientes. Em cada iteração, o modelo é ajustado para corrigir os erros cometidos pelas iterações anteriores, com base nos gradientes da função de perda.

1.1.3. Árvores de Decisão como Estimadores Base:

Por padrão, o `GradientBoostingClassifier` utiliza árvores de decisão como estimadores base. Cada árvore é ajustada sequencialmente para corrigir os erros residuais do conjunto anterior de árvores.

1.1.4. Treinamento Sequencial:

O treinamento do modelo é feito de maneira sequencial. Cada árvore é treinada para corrigir os erros do conjunto anterior de árvores. Isso é feito através da atualização dos pesos das instâncias no conjunto de treinamento, dando mais importância às instâncias que foram classificadas incorretamente.

1.1.5. Parâmetros Ajustáveis:

O `GradientBoostingClassifier` possui vários parâmetros ajustáveis, incluindo a taxa de aprendizado (`learning_rate`), o número de árvores (`n_estimators`), a profundidade máxima das árvores (`max_depth`), entre outros. Esses parâmetros permitem ajustar o desempenho e a complexidade do modelo.

1.1.6. Regularização:

O algoritmo pode incluir técnicas de regularização para evitar overfitting. Por exemplo, a profundidade máxima das árvores pode ser limitada para controlar a complexidade do modelo.

1.1.7. Desvantagens:

Enquanto o Gradient Boosting é poderoso e geralmente produz modelos precisos, ele pode ser sensível a outliers e pode exigir mais ajuste de hiperparâmetros em comparação com outros algoritmos. Além disso, como o treinamento é sequencial, o processo pode levar mais tempo.

1.1.8. Uso Geral:

O `GradientBoostingClassifier` é amplamente utilizado em competições de ciência de dados e em aplicações práticas devido à sua eficácia em muitos tipos de conjuntos de dados.

1.2. KNeighborsClassifier:

O `KNeighborsClassifier` é um algoritmo de classificação que pertence à família de métodos conhecidos como "k-nearest neighbors" (k-NN), ou vizinhos mais próximos. Esse algoritmo é utilizado para tarefas de classificação em aprendizado de máquina e está disponível na biblioteca scikit-learn em Python.

Aqui estão alguns pontos-chave sobre o `KNeighborsClassifier`:

1.2.1. Baseado em Vizinhos Mais Próximos:

O algoritmo K-NN é do tipo "instance-based", o que significa que ele não constrói um modelo explícito durante o treinamento. Em vez disso, ele armazena

todas as instâncias do conjunto de treinamento e faz previsões com base na proximidade dessas instâncias a uma nova instância de entrada.

1.2.2. Parâmetro K:

O "k" em K-NN representa o número de vizinhos mais próximos a serem considerados ao fazer uma previsão. Durante o treinamento, o modelo armazena todas as instâncias do conjunto de treinamento. Quando uma previsão é solicitada, o modelo identifica os k vizinhos mais próximos da instância de entrada e a classe mais frequente entre esses vizinhos é atribuída à instância de entrada.

1.2.3. Distâncias:

A métrica de distância é crucial no K-NN, pois determina como a proximidade entre instâncias é medida. A distância Euclidiana é comumente usada, mas outras métricas, como a distância de Manhattan, também podem ser empregadas, dependendo do problema.

1.2.4. Ajuste aos Dados:

O K-NN é um algoritmo não paramétrico, o que significa que não faz suposições explícitas sobre a forma funcional dos dados. Ele se ajusta diretamente aos dados de treinamento.

1.2.5. Sensível à Escala:

O K-NN é sensível à escala das variáveis. Portanto, normalmente é aconselhável escalar os dados antes de aplicar o K-NN para garantir que todas as variáveis tenham a mesma influência na determinação da proximidade.

1.2.6. Uso em Classificação:

O `KNeighborsClassifier` é comumente usado para problemas de classificação, onde o objetivo é prever a classe de uma instância com base nas classes das instâncias vizinhas mais próximas no espaço de características.

1.2.7. Uso em Regressão:

Além de classificação, a abordagem de k-NN também pode ser estendida para problemas de regressão, sendo chamada de "k-nearest neighbors regression" ou KNR. Nesse caso, a previsão é a média (ou outra medida) dos valores das instâncias vizinhas mais próximas.

1.3. LightGBMClassifier

O LightGBM (Light Gradient Boosting Machine) é um framework de boosting baseado em gradientes, projetado para treinamento distribuído e eficiente de modelos de machine learning em larga escala. Ele foi desenvolvido pela Microsoft e é de código aberto. O LightGBM é especialmente conhecido por sua rapidez e eficiência, sendo adequado para lidar com conjuntos de dados grandes e espaços de características de alta dimensão.

Aqui estão algumas características e conceitos-chave associados ao LightGBM:

1.3.1. Framework de Boosting Baseado em Gradientes:

O LightGBM é baseado no framework de boosting baseado em gradientes, uma técnica de aprendizado de conjunto. Ele constrói um modelo preditivo na forma de um conjunto de "weak learners" (geralmente árvores de decisão) para criar um "strong learner".

1.3.2. Algoritmos de Aprendizado Baseados em Árvores:

O LightGBM utiliza algoritmos de aprendizado baseados em árvores para tarefas de classificação e regressão. Ele constrói árvores de forma "leaf-wise" (ou seja, crescimento folha por folha), o que pode levar a uma convergência mais rápida e a uma utilização de memória reduzida.

1.3.3. Leve e Eficiente:

Uma das principais vantagens do LightGBM é sua eficiência. Ele é projetado para ser rápido e requer menos memória em comparação com outras implementações de boosting. Isso o torna adequado para conjuntos de dados grandes e espaços de características de alta dimensão.

1.3.4. Crescimento de Árvore Folha por Folha:

O LightGBM cresce as árvores de forma "leaf-wise" (folha por folha), em oposição à abordagem "level-wise". Nessa estratégia, o algoritmo escolhe a folha com o máximo ganho de informação (melhoria na função de perda) para expandir, o que pode resultar em um modelo mais preciso com menos folhas.

1.3.5. Amostragem Baseada em Gradiente de Um Lado (GOSS) e Agrupamento Exclusivo de Recursos (EFB):

O LightGBM incorpora técnicas como a Amostragem Baseada em Gradiente de Um Lado (GOSS) e o Agrupamento Exclusivo de Recursos (EFB) para melhorar ainda mais a velocidade de treinamento e reduzir o uso de memória. O GOSS é um método de subamostragem de dados que se concentra em instâncias com gradientes grandes, e o EFB agrupa recursos exclusivos para reduzir o número de recursos usados no processo de treinamento.

1.3.6. Suporte a Características Categóricas:

O LightGBM oferece suporte nativo para características categóricas. Ele pode lidar com variáveis categóricas sem a necessidade de codificação one-hot, o que pode ser vantajoso em termos de eficiência computacional e desempenho do modelo.

1.3.7. Computação Distribuída:

O LightGBM suporta treinamento distribuído, permitindo a paralelização do processo de aprendizado em várias máquinas, o que pode reduzir significativamente os tempos de treinamento para conjuntos de dados grandes.

1.3.8. Validação Cruzada e Parada Antecipada:

O LightGBM suporta validação cruzada k-fold para avaliar o desempenho do modelo. Ele também inclui funcionalidade de parada antecipada, permitindo que o treinamento seja interrompido quando o desempenho em um conjunto de validação para de melhorar, ajudando a evitar a sobreajustagem.

Em resumo, o LightGBM é um framework de boosting poderoso e eficiente que se destaca no manuseio de conjuntos de dados grandes e espaços de características de alta dimensão. Sua rapidez, eficiência de memória e capacidades de computação distribuída o tornam uma escolha popular para uma ampla variedade de tarefas de machine learning.

1.4. RandomForestClassifier

O `RandomForestClassifier` é um algoritmo de aprendizado de máquina que pertence à categoria de métodos ensemble, mais especificamente à família de Random Forests. Ele é utilizado para tarefas de classificação e faz parte da biblioteca scikit-learn em Python.

Aqui estão alguns pontos-chave sobre o `RandomForestClassifier`:

1.4.1. Ensemble Learning:

O `RandomForestClassifier` é uma implementação de ensemble learning, que combina a predição de vários estimadores (geralmente árvores de decisão) para melhorar a robustez e o desempenho do modelo.

1.4.2. Árvores de Decisão como Estimadores Base:

Cada estimador no conjunto é uma árvore de decisão. No entanto, ao contrário de uma única árvore de decisão, o Random Forest utiliza várias árvores independentes, treinadas com diferentes subconjuntos do conjunto de treinamento.

1.4.3. Amostragem Aleatória de Dados e Features:

O nome "Random Forest" refere-se à aleatoriedade introduzida no processo de construção de cada árvore. Durante o treinamento de cada árvore, uma amostra aleatória (com substituição) é retirada do conjunto de treinamento (bootstrapping). Além disso, em cada divisão de um nó, um subconjunto aleatório de features é considerado.

1.4.4. Votação por Maioria:

No momento de fazer uma previsão, cada árvore no conjunto vota em uma classe, e a classe mais frequente entre as árvores é escolhida como a previsão final do Random Forest.

1.4.5. Redução de Overfitting:

A aleatoriedade introduzida pelo processo de amostragem e pela seleção de features ajuda a reduzir o overfitting, tornando o modelo mais geral e capaz de lidar com uma variedade de padrões nos dados.

1.4.6. Paralelismo:

A construção de cada árvore na floresta é independente das outras, o que permite treinar árvores em paralelo, proporcionando uma melhoria significativa no tempo de treinamento em comparação com uma única árvore de decisão.

1.4.7. Parâmetros Ajustáveis:

O `RandomForestClassifier` possui vários parâmetros ajustáveis, como o número de árvores na floresta (`n_estimators`), a profundidade máxima das árvores (`max_depth`), entre outros. Esses parâmetros podem ser ajustados para otimizar o desempenho do modelo.

1.4.8. Uso em Classificação:

O `RandomForestClassifier` é comumente usado para problemas de classificação, onde o objetivo é prever a classe de uma instância com base nas decisões de várias árvores de decisão.

2. ANÁLISE EXPLORATÓRIA DE DADOS

A base de dados possui mais de 1.2 milhão de entradas e 23 features; serão aplicadas técnicas de amostragem e feature selection para fins de otimização do modelo.

As features da base são:

trans_date_trans_time: referem-se ao horário do dia em que ocorreu a transação;

cc_num: número do cartão de crédito;

merchant: estabelecimento em que houve a transação;

category: categoria do estabelecimento/transação (entretenimento, viagens, mercado, gás);

amt: valor da transação financeira realizada;

first: primeiro nome do titular do cartão de crédito;

last: último nome do titular do cartão;

gender: gênero do titular do cartão;

street: endereço de rua do titular do cartão;

city: cidade de residência do titular do cartão;

state: estado de residência do titular do cartão;

zip: código postal do titular do cartão;

lat: localização latitudinal do titular do cartão;

long: localização longitudinal do titular do cartão;

city_pop: população da cidade do titular do cartão;

job: emprego/ocupação do titular do cartão;

dob: dia de nascimento do titular do cartão;

trans_num: número da transação financeira;

unix_time: horário da transação pelo timestamp UNIX;

merch_lat: localização latitudinal do estabelecimento;

merch_long: localização longitudinal do estabelecimento;

is_fraud: marcador de transação fraudulenta (target class).

Na figura abaixo pode-se observar a distribuição de transações fraudulentas e não fraudulentas “is_fraud” em relação ao valor das transações realizadas:

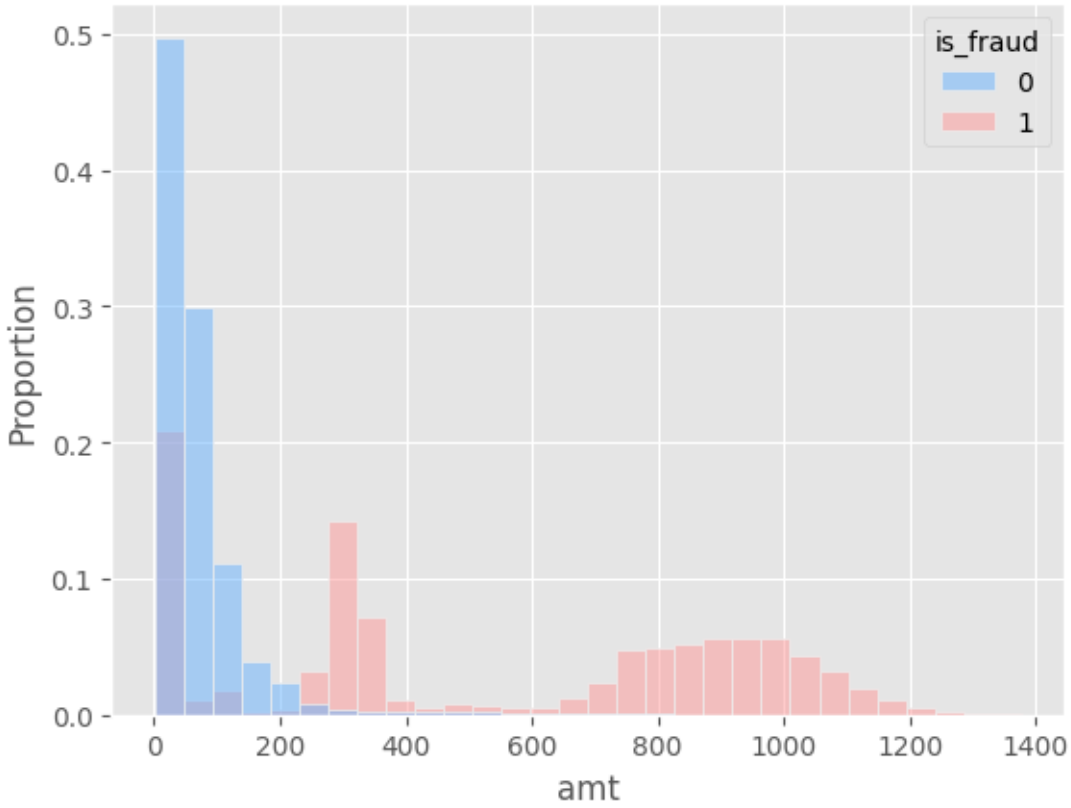

Figura 2.1 – Proporção de transações fraudulentas e não fraudulentas em relação ao valor gasto.

Nota-se uma grande diferença entre a distribuição dessas transações sendo a concentração de transações não fraudulentas bastante concentrada entre 0 – 200 USD e uma distribuição mais larga para transações fraudulentas em 3 blocos de 0 – 100, 200 – 400 e 600 – 1200 USD.

A figura a seguir nos mostrará a distribuição dessas transações entre as categorias de estabelecimento ou transação:

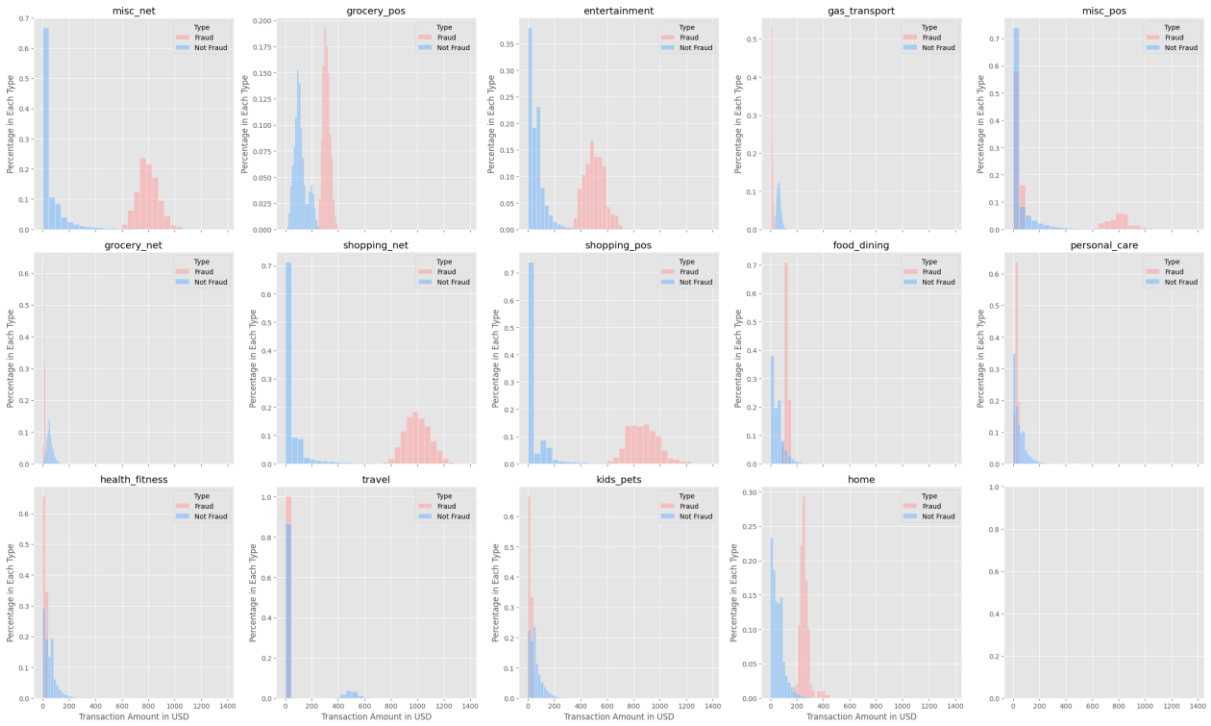

Figura 2.2 - Proporção de transações fraudulentas e não fraudulentas em relação ao valor gasto por categoria.

Nota-se que na grande maioria das categorias as transações demonstram um comportamento diferente na target class com transações fraudulentas com valores mais elevados do que as comuns.

Ao olharmos para a distribuição de fraudes entre as categorias notamos um maior volume na categoria “grocery_pos” e menor em “travel”:

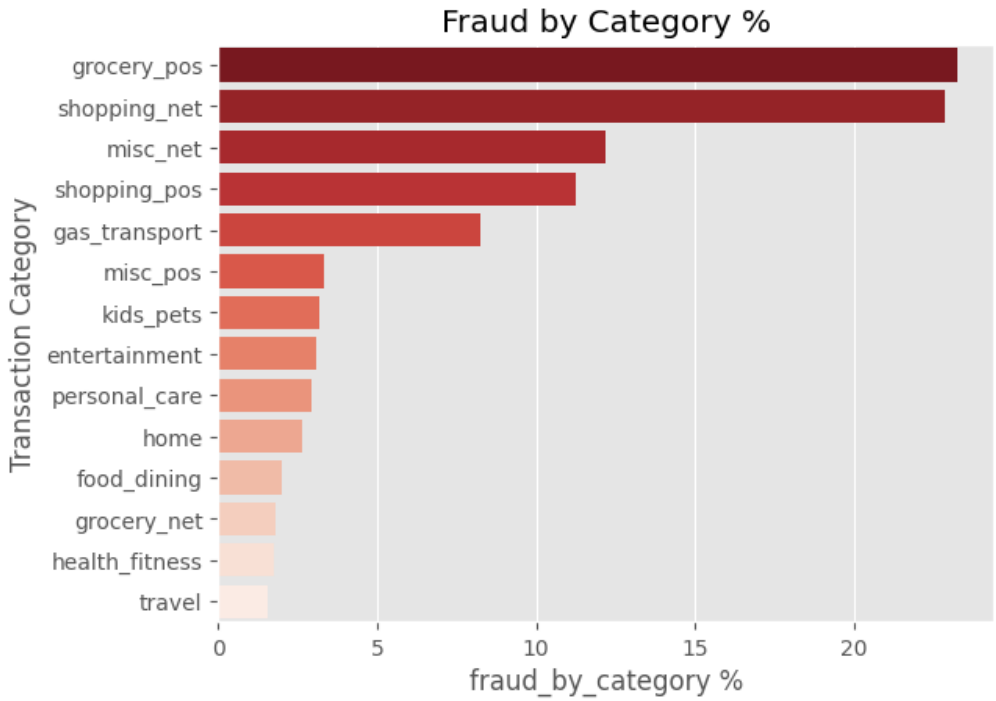

Figura 2.3 – Proporção de fraudes por categoria.

Entretanto aocomparar o volume de transações agrupado pela target class em cada categoria pode-se observar qual categoria tem maior probabilidade de receber uma fraude. Observa abaixo:

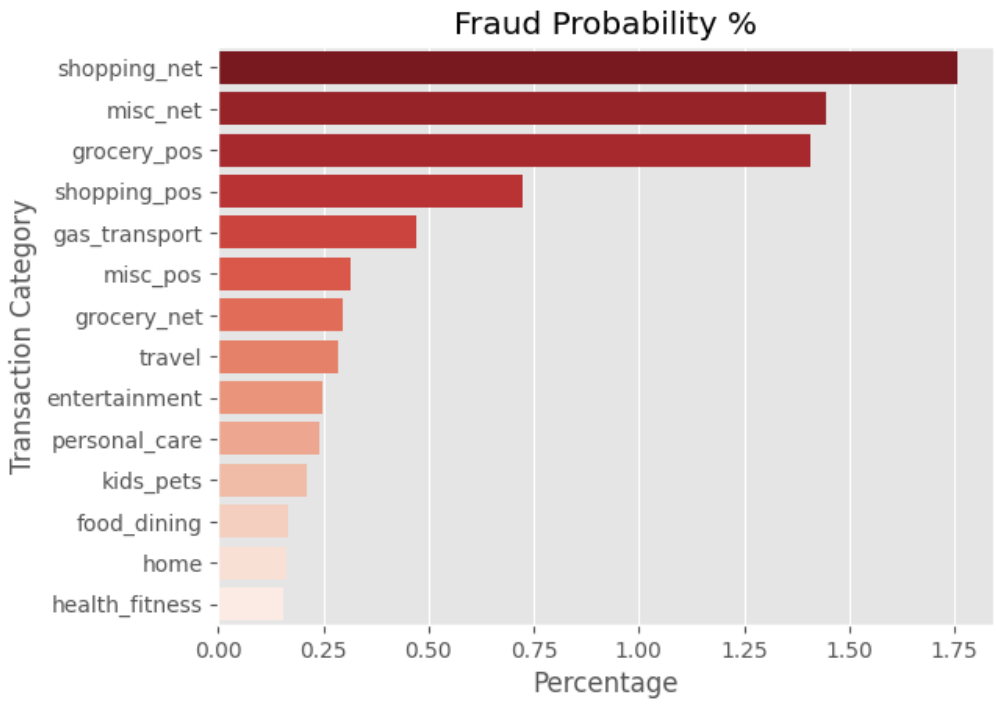

Figura 2.4 – Probabilidade de fraude por categoria.

A categoria com maior probabilidade de ocorrer uma fraude é shopping_net com 1,75% e a menor é health_fitness com menos de 0,25%.

Outro indicador que vale se destacar é comportamento das transações ao longo das horas do dia, como observado abaixo:

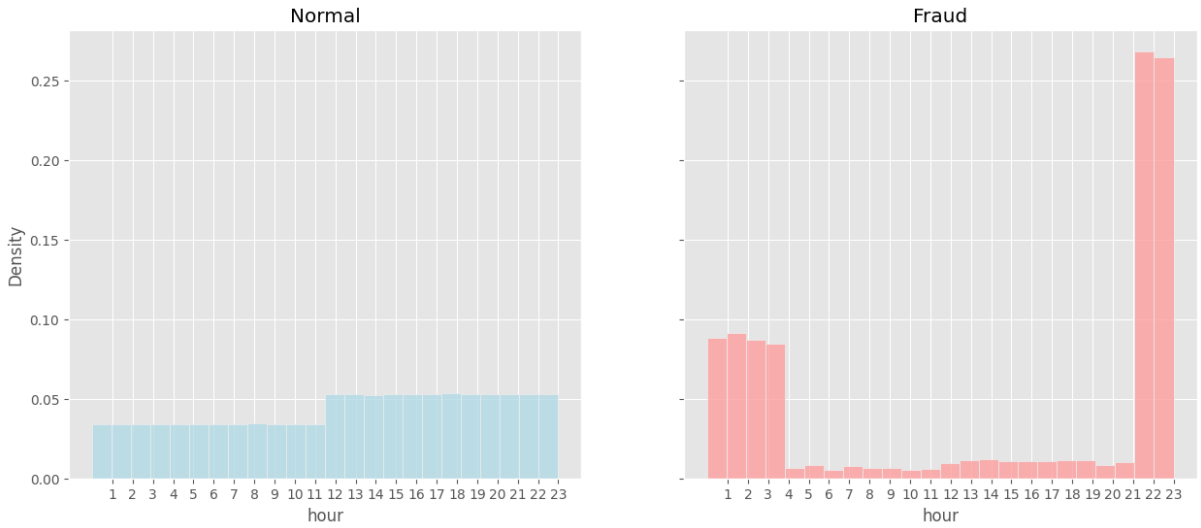

Figura 2.5 – Comportamento das transações ao longo das horas do dia.

É possível ver uma grande concentração de fraudes cometidas entre 21h00 – 23h00 e depois um volume considerável entre 00h00 – 04h00. Quanto as transações não fraudulentas, nota-se um aumento no volume entre 12h00 – 23h00, entretanto apresenta um comportamento mais constante.

A frequência de uso dos carões de crédito também nos mostra uma análise interessante:

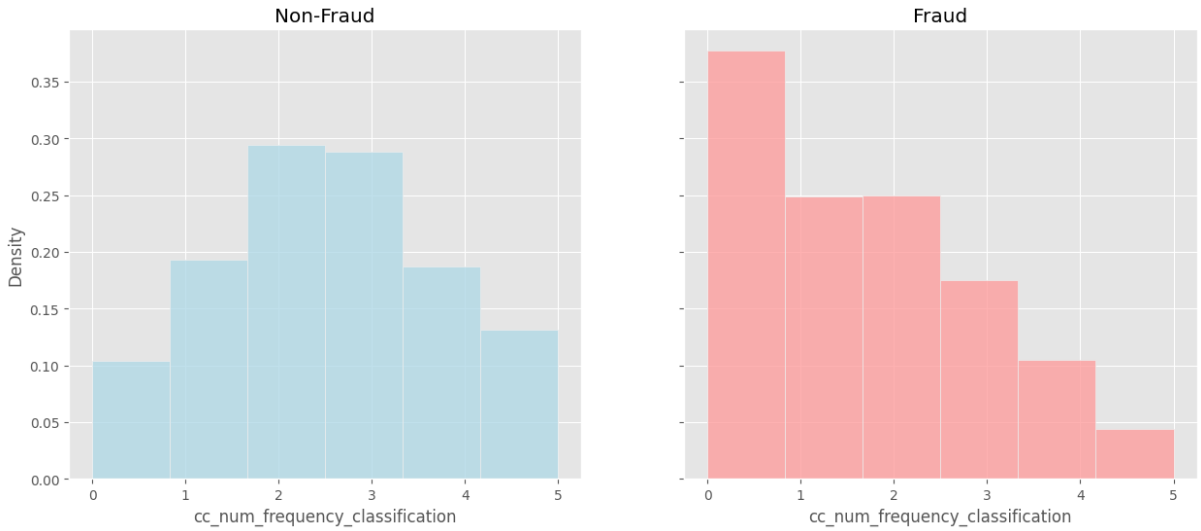

Figura 2.5 – Frequência de uso dos cartões de crédito.

Observa-se que a maioria das transações fraudulentas ocorrem com cartões utilizados uma única vez e decresce bastante o número de fraudes com uma frequência de uso maior.

3. PREPARAÇÃO DA BASE DE DADOS

Inicialmente, por se tratar de uma modelagem supervisionada com objetivo de classificação do evento fraude, vamos utilizar a regressão logística.

Ainda, por tratar-se de uma base desbalanceada em que a incidência do evento monitorado corresponde a aproximadamente 0.6% dos casos, será aplicada uma técnica de transformação conhecida por undersampling para obter uma proporção que implique no melhor funcionamento do modelo. A proporção escolhida foi 9/1. Como mostra a figura a seguir:

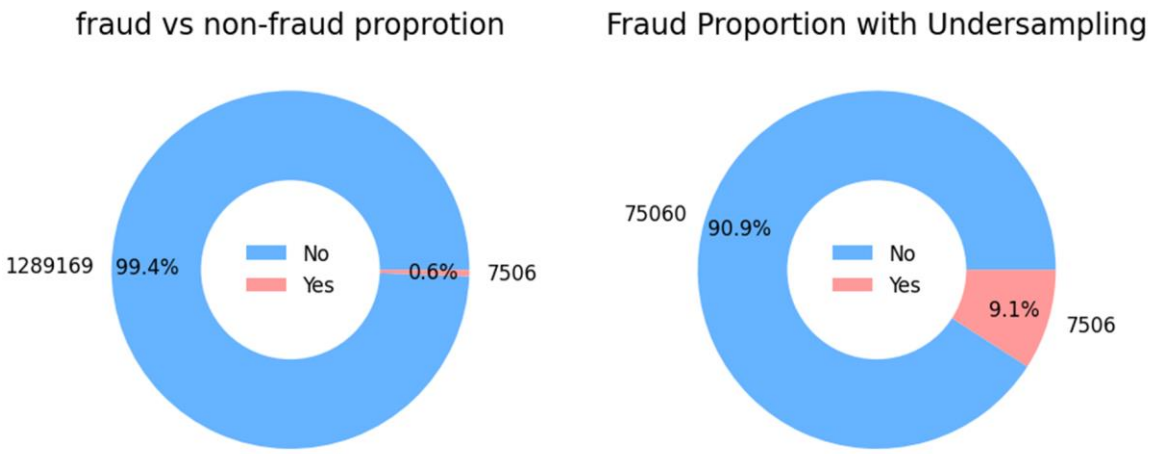

Figura 2.1 – Proporções da amostra antes e depois do undersampling.

Com relação ao tratamento das variáveis da base, utilizou-se técnicas de categorical encoding para features categóricas, para o processo de feature selection, aplicaremos WOE e boruta, dentre outras que se mostrarem necessárias.

WOE (Weight of Evidence) é uma técnica utilizada em modelagem estatística e de risco de crédito para transformar variáveis independentes (preditoras) de maneira a serem mais informativas em relação à variável dependente (a variável que se deseja prever). Essa técnica é frequentemente aplicada em modelos de pontuação de crédito e em problemas nos quais a predição da probabilidade é crucial.

A ideia central por trás do WOE é calcular o logaritmo da razão entre a probabilidade de evento (por exemplo, inadimplência em um empréstimo) e a probabilidade de não evento para cada categoria de uma variável. Em outras palavras, o WOE é uma medida da força de uma variável em prever o evento desejado.

A fórmula geral para o cálculo do WOE para uma categoria específica:

$$WOE_i = \ln\left(\frac{P(\text{Evento em } i)}{P(\text{Não Evento em } i)}\right)$$

onde $P(\text{Evento em } i)$ é a probabilidade de ocorrer o evento na categoria i, e $P(\text{Não Evento em } i)$ é a probabilidade de não ocorrer o evento na categoria i.

4. MÉTRICAS DE AVALIAÇÃO DOS MODELOS

Para monitorar o desempenho do modelo, serão analisadas as métricas: acurácia, precision, recall, F1-score, ROC e AUC.

Accuracy (Acurácia):

A acurácia é uma métrica simples que mede a proporção de instâncias corretamente classificadas em relação ao total de instâncias. A fórmula é dada por:

$$\text{Acurácia} = \frac{\text{Número de previsões corretas}}{\text{Total de instâncias}}$$

Embora seja uma métrica fácil de entender, a acurácia pode ser enganosa em problemas com classes desbalanceadas.

Precision (Precisão):

A precisão é a proporção de instâncias positivas previstas corretamente em relação ao total de instâncias positivas previstas. A fórmula é dada por:

$$\text{Precision} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Positivos}}$$

A precisão é útil quando o custo de falsos positivos é alto.

Recall (Revocação ou Sensibilidade):

O recall é a proporção de instâncias positivas previstas corretamente em relação ao total de instâncias positivas reais. A fórmula é dada por:

$$\text{Recall} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Negativos}}$$

O recall é útil quando o custo de falsos negativos é alto.

F1-score:

O F1-score é uma métrica que combina precision e recall em uma única pontuação. É a média harmônica entre precision e recall. A fórmula é dada por:

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

O F1-score é útil quando há um equilíbrio desejado entre precision e recall.

ROC (Receiver Operating Characteristic) e AUC (Area Under the Curve):

A curva ROC é uma representação gráfica da taxa de verdadeiros positivos em função da taxa de falsos positivos para diferentes valores de limiar de classificação. O AUC é a área sob a curva ROC e fornece uma medida da capacidade discriminativa do modelo, variando de 0 a 1. Quanto maior o AUC, melhor o desempenho do modelo em distinguir entre classes.

Uma curva ROC próxima do canto superior esquerdo indica um bom desempenho, enquanto uma curva que segue a diagonal é indicativa de um desempenho aleatório. Como o exemplo na figura abaixo:

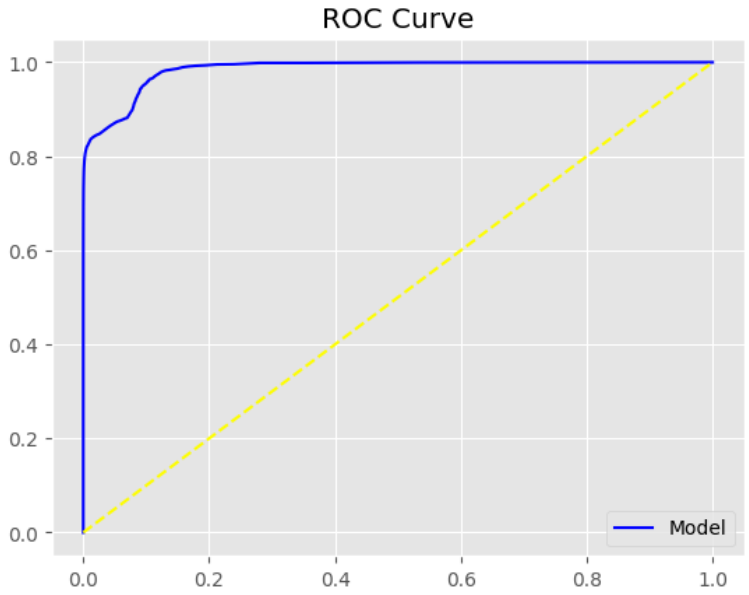

Figura 4.1 – Exemplo de curva ROC.

A escolha dessas métricas de monitoramento complementares à acurácia se deve ao fato de não ser uma métrica de desempenho suficientemente satisfatória para análise de modelos cujas bases sejam altamente desbalanceadas, como a do presente estudo.

5. VERIFICAÇÃO DE NORMALIDADE DA AMOSTRA

A fim de evitar algum viés que possa interferir na tomada de decisão dos modelos foi verificada a normalidade da nossa feature “amt” que nos mostra o valor de cada transação realizada, como mostrado na figura abaixo:

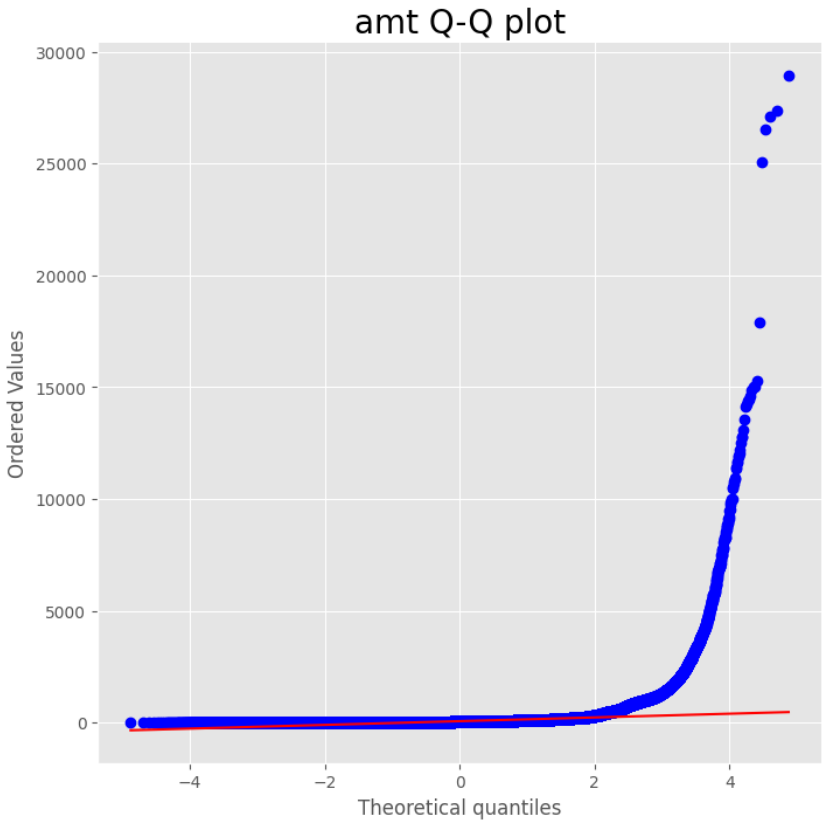

Figura 5.1 – Verificação de normalidade de “amt”.

Nota-se que a distribuição está longe de demonstrar um comportamento normal pois os pontos azuis não seguem o comportamento da reta em vermelho e isso pode afetar a tomada de decisão dos modelos. Então para aproximar o comportamento de “amt” de uma curva normal foi criada uma nova feature “amt_log” onde foi criada uma escala logarítmica que que diminui a incidência de outliers devido ao encurtamento do “range” entre os dados como mostrado abaixo:

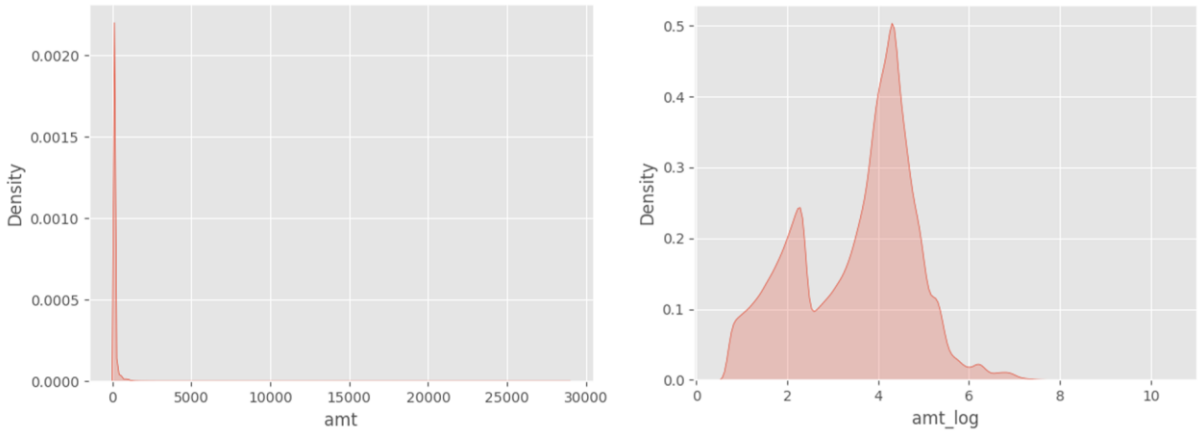

Figura 5.2 – Curva de densidade de “amt” vs “amt_log”

Enquanto a feature “amt” varia de 0 – 30000 USD, a escala logarítmica “amt_log” passa a variar entre 0 – 10 e consequentemente seu comportamento fica seu comportamento fica mais próximo de uma normal como pode-se observar abaixo:

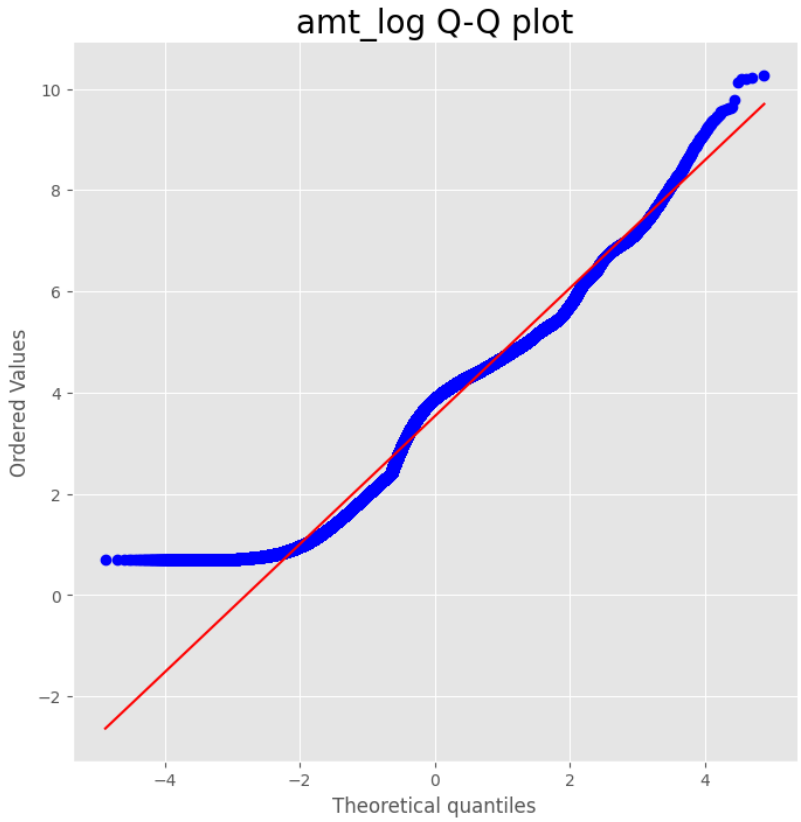

Figura 5.3 – Curva de densidade de “amt” vs “amt_log”

6. FEATURES IMPORTANCE

As duas bases de dados utilizadas neste estudo são robustas e contém bastante informação sendo a base te treinamento dos modelos uma tabela com 23 coluna e mais de 1,2 milhão de linhas e a de teste e validação dos modelos com mais de 0,5 milhão.

Então para assegurar que os modelos sejam treinados apenas utilizando features relevantes, foram criados 2 vetores aleatórios chamados de “random_feat_1” e “random_feat_2” e utilizamos o algoritmo RandomForestClassifier para verificar o peso de cada feature na tomada de decisão do modelo. O resultado é visto na figura a seguir:

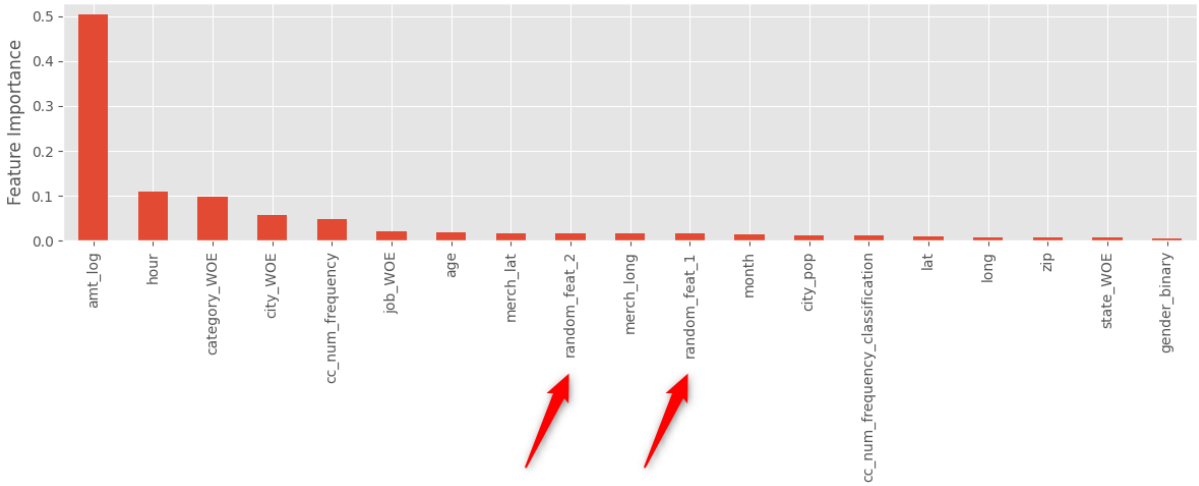

Figura 6.1 – Features importance.

Então foram desconsideradas todas as features que mostraram menor importância na tomada de decisão do modelo comparadas aos vetores aleatórios.

7. RESULTADO DOS MODELOS PREDITIVOS

A seguir pode se observar o desempenho dos modelos escolhidos para o estudo. Serão analisadas as métricas: accuracy, precision, recall, F1-score, ROC e AUC juntamente com a matriz de confusão.

7.1. GradientBoostingClassifier:

Métricas obtidas:

Métricas	Treino	Teste
Accuracy	0,98	0,97
Precision	0,95	0,11
Recall	0,86	0,92
F1 Score	0,91	0,19
AUC	0,99	0,98

Tabela 7.1 – Métricas obtidas com GradientBoostingClassifier.

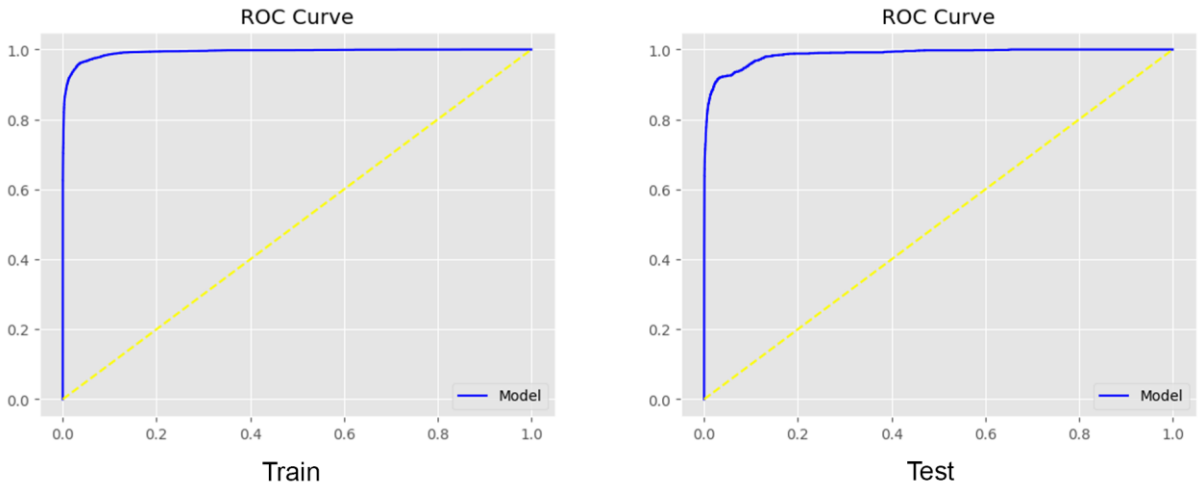

Figura 7.1 – Curva ROC com GradientBoostingClassifier.

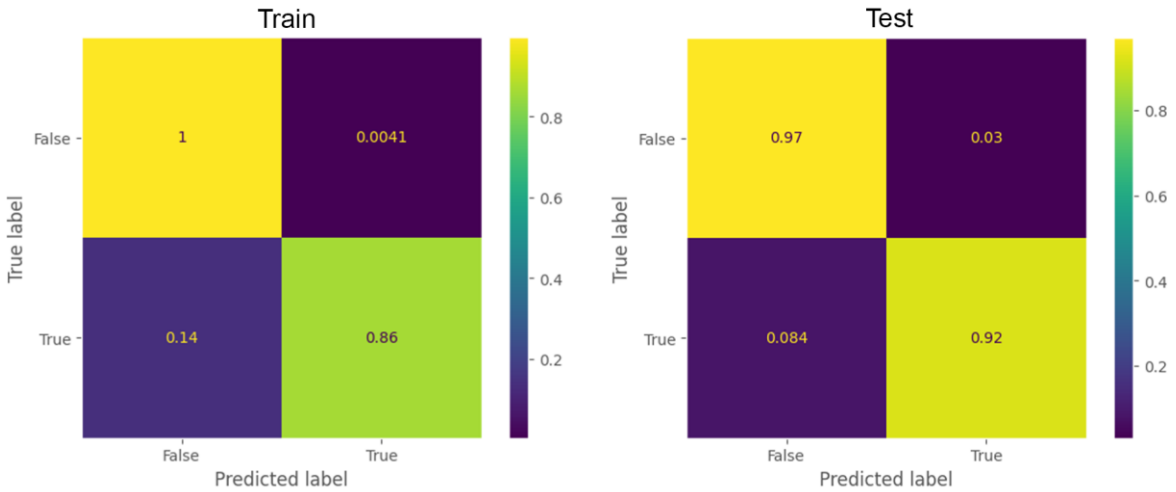

Figura 7.2 – Matriz de confusão com GradientBoostingClassifier.

7.2. KNeighborsClassifier:

Métricas obtidas:

Métricas	Treino	Teste
Accuracy	0,96	0,86
Precision	0,93	0,01
Recall	0,61	0,48
F1 Score	0,74	0,03
AUC	0,98	0,70

Tabela 7.2 – Métricas obtidas com KNeighborsClassifier.

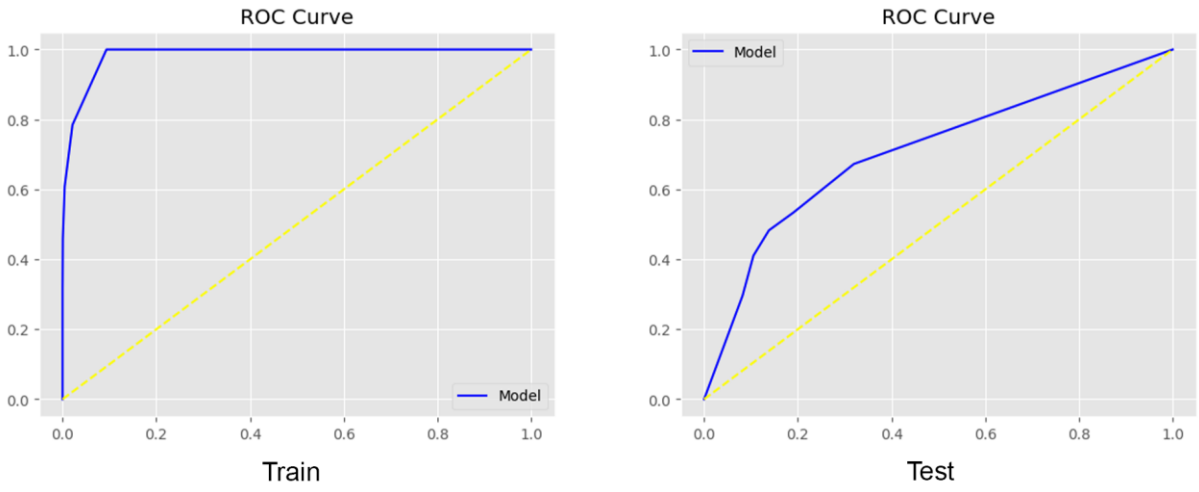

Figura 7.3 – Curva ROC com KNeighborsClassifier.

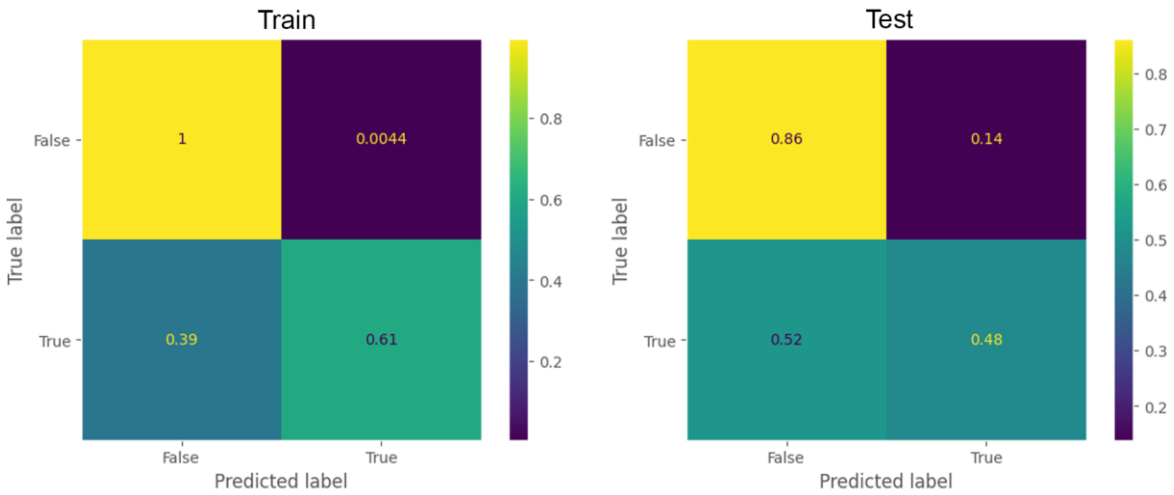

Figura 7.4 – Matriz de confusão com KNeighborsClassifier.

7.3. LightGBMClassifier:

Métricas obtidas:

Métricas	Treino	Teste
Accuracy	1,00	0,99
Precision	0,99	0,29
Recall	0,99	0,89
F1 Score	0,99	0,43
AUC	0,99	0,99

Tabela 7.3 – Métricas obtidas com LightGBMClassifier.

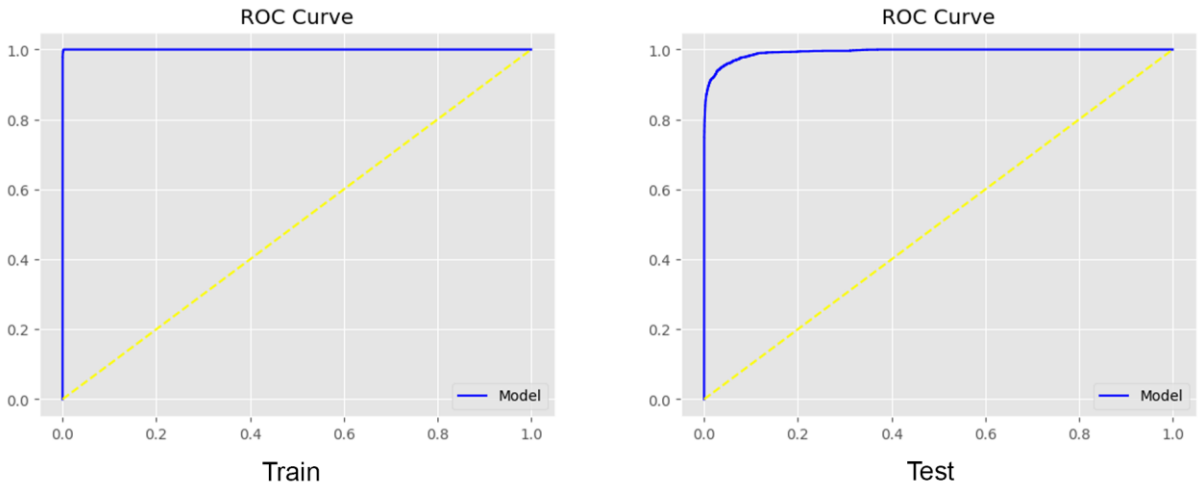

Figura 7.5 – Curva ROC com LightGBMClassifier.

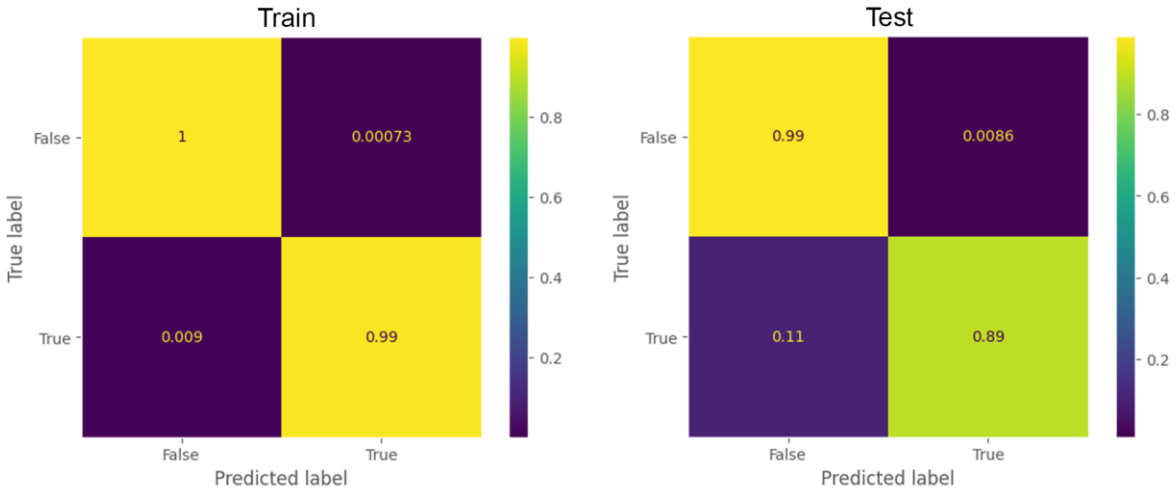

Figura 7.6 – Matriz de confusão com LightGBMClassifier.

7.4. RandomForestClassifier:

Métricas obtidas:

Métricas	Treino	Teste
Accuracy	1,00	0,92
Precision	1,00	0,04
Recall	1,00	0,92
F1 Score	1,00	0,08
AUC	1,00	0,98

Tabela 7.4 – Métricas obtidas com RandomForestClassifier.

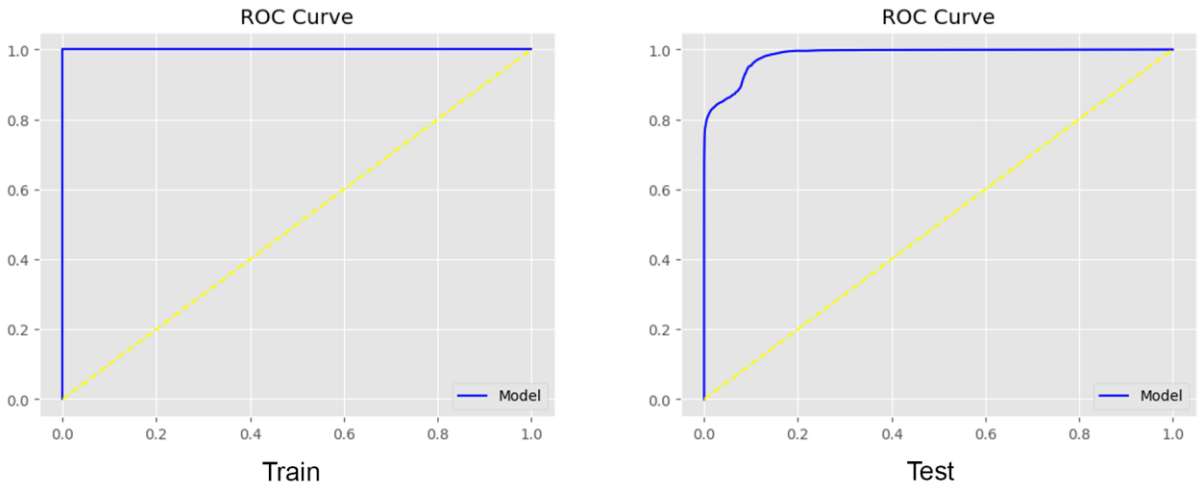

Figura 7.7 – Curva ROC com RandomForestClassifier.

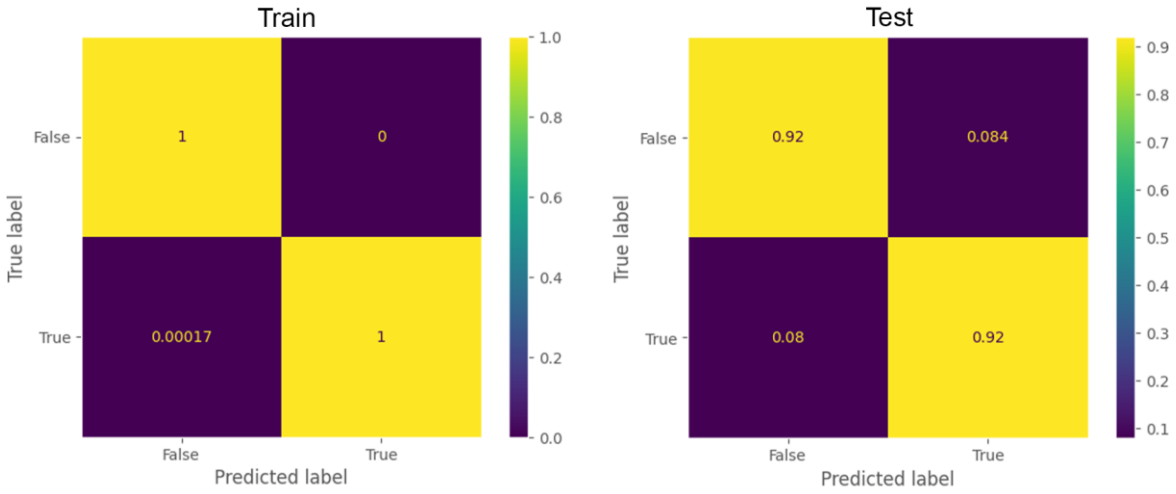

Figura 7.8 – Matriz de confusão com RandomForestClassifier.

8. CONCLUSÃO

Ao analisar as métricas dos modelos obtidas na base de teste o modelo que mostra um melhor desempenho na detecção de fraudes é o modelo LightGBMClassifier como mostra a tabela a seguir:

métricas	gboost	knn	lgbm	rf
Accuracy	0,97	0,86	0,99	0,92
Precision	0,11	0,01	0,29	0,04
Recall	0,92	0,48	0,89	0,92
F1 Score	0,19	0,03	0,43	0,08
AUC	0,98	0,70	0,99	0,98

Tabela 8.1 – Métricas obtidas com a base de teste.

Devido ao desbalanceamento das bases de dados analisadas a métrica de acuracy torna-se irrelevante visto que essa métrica permaneceria alta mesmo se errasse 100% dos casos de fraude.

Quanto a precision ficou em 29% indicando um grande volume de transações normais consideradas como fraude, porém o recall em 89% mostra que em contrapartida poucas fraudes passaram pelo modelo como transações normais. Consequentemente o F1-score do LightGBMClassifier se destacou em relação aos outros modelos com 43%.

O modelo também mostrou uma boa otimização da curva ROC fazendo com que ela ficasse bem próximo do valor 1 em sua extremidade indicando um alto índice de acertos em nosso modelo como mostra a figura abaixo:

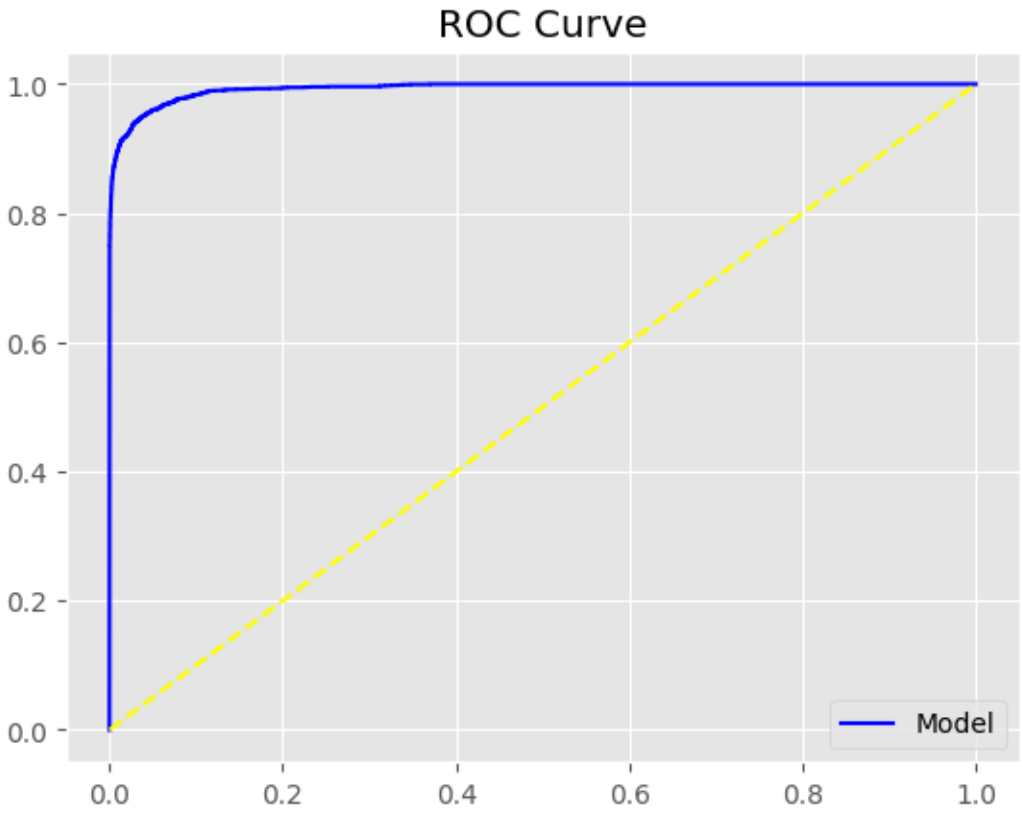

Figura 8.1 – Curva ROC do modelo LightGBMClassifier na base de teste.

9. REFERÊNCIAS

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

https://neptune.ai/blog/gradient-boosted-decision-trees-guide

https://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

https://lightgbm.readthedocs.io/en/stable/

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://www.sas.com/pt_br/insights/analytics/analises-preditivas.html

https://cloud.google.com/learn/what-is-predictive-analytics?hl=pt-br