UNIVERSIDADE DE BRASÍLIA

Faculdade do Gama

Aprendizado de Máquina

Mini Trabalho 5

Seleção de modelos com potencial

Grupo - 1

Ana Clara Barbosa Borges

André Emanuel Bispo da Silva

Artur Handow Krauspenhar

Gabriel Moura dos Santos

João Artur Leles Ferreira Pinheiro

João Pedro Anacleto Ferreira Machado

Brasília, DF

2025

Introdução

Primeiramente, para garantir que todos os testes fossem realizados sobre a mesma base de comparação, os dados foram divididos previamente, utilizando 80% da amostra para treinamento e os 20% restantes para teste. Essa divisão foi realizada por meio da função train_test_split da biblioteca Scikit-learn, e os conjuntos foram salvos nos arquivos train.csv e test.csv. Essa divisão fixa assegura que todos os modelos avaliados utilizem exatamente os mesmos conjuntos de dados, o que visa garantir uma comparação justa e consistente entre os resultados.

Modelos testados

KNN

Foi testado o modelo classificatório K Nearest Neighbours, um modelo que classifica pontos baseado em um voto de maioria entre os K-ésimos pontos de referência mais próximos. Dessa forma foi feita uma busca de hiperparâmetros ótimos para o modelo na máquina “hu1” da chococino, apresentando uma cpu Intel(R) Xeon(R) CPU E5-2680 v3 de 2.50GHz com 48 núcleos e 2 threads virtuais por núcelo, permitindo o processamento de várias instâncias de teste em paralelo e aproximadamente 800 Gb de memória RAM.

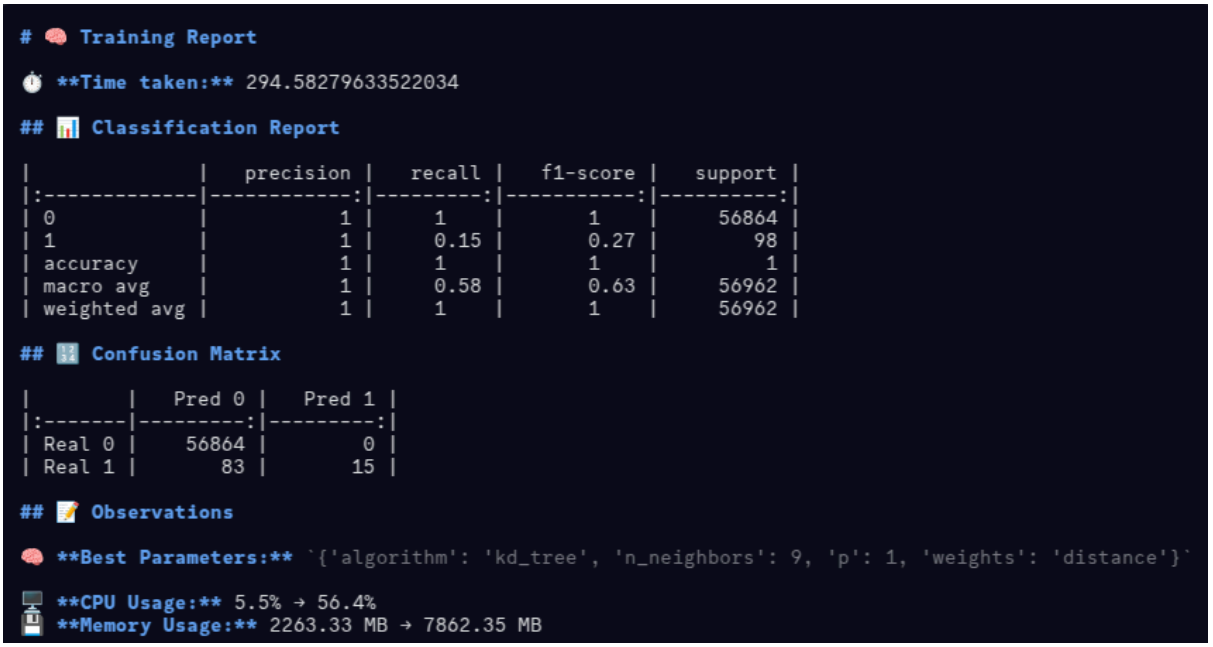
Os parâmetros escolhidos para o modelo foram:

- K: número de vizinhos a serem contandos, valor escolhido aleatoriamente entre 1 e 20
- p: parâmetro de potência da métrica de minkowski, restrito à 1 e 2 (distância manhattan e distância euclidiana respectivamente)
- pesos: uniformes ou inverso da distância
- algoritmo: um de: escolha automática, Ball Tree, K-D Tree e força bruta

Os parâmetros escolhidos para a busca foram:

- 20 iterações
- Validação cruzada com 5 folds
- número de jobs equivalente ao número de cpus

Além disso, foram coletadas métricas sobre uso de cpu, tempo de processamento e uso de memória durante a execução dos testes.



🧠 Training Report

⏱️ **Time taken:** 294.58279633522034

📊 Classification Report

	precision	recall	f1-score	support
0	1	1	1	56864
1	1	0.15	0.27	98
accuracy	1	1	1	1
macro avg	1	0.58	0.63	56962
weighted avg	1	1	1	56962

🔢 Confusion Matrix

	Pred 0	Pred 1
Real 0	56864	0
Real 1	83	15

📝 Observations

🧠 **Best Parameters:** `{'algorithm': 'kd_tree', 'n_neighbors': 9, 'p': 1, 'weights': 'distance'}`

🖥️ **CPU Usage:** 5.5% → 56.4%
🧠 **Memory Usage:** 2263.33 MB → 7862.35 MB

Relatório gerado em markdown pelo programa

Os resultados não foram promissores mostrando um recall de 15% e F1 score de 27% para transações fraudulentas, experimentos feitos anteriormente manualmente corroboram estes resultados. Além disso, necessitou de aproximadamente 5 minutos, 7,8Gb de memória ram e 56,4% de uso de CPU para efetuar todos os testes.

Assim, decidimos não utilizar o KNN por si só nem como participante em um modelo composto.

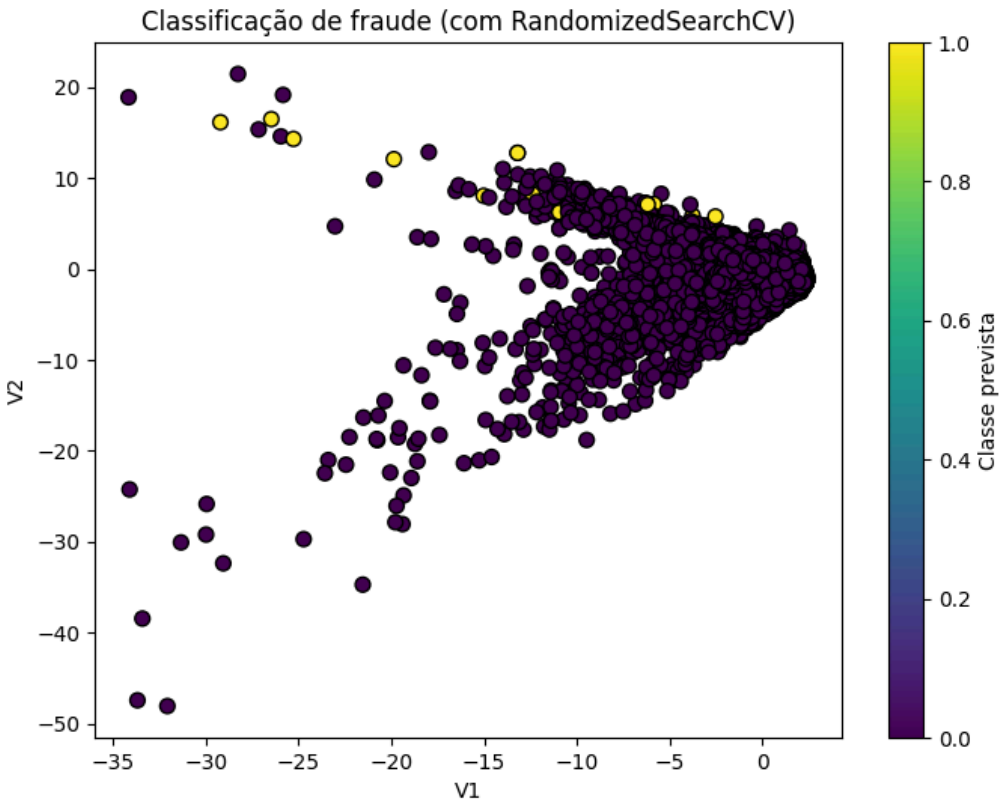


Gráfico relacionando as features V1 e V2 e o resultado final da classificação

O relatório completo sobre os testes e o modelo ótimo podem ser encontrados na pasta onde se encontra este documento.

XGBoost

O modelo XGBoost foi selecionado por ser um algoritmo eficiente para problemas de classificação, uma vez que é capaz de lidar bem com desbalanceamento de classes (comum em problemas de detecção de fraude) e capturar relações complexas nos dados.Alguns do seus principais parâmetros como:

- **n_estimators**(Número de árvores no modelo) = **5**
- **max_depth** (Profundidade máxima das árvores)= **4**
- **learning_rate** (Taxa de aprendizado)= **0.5**
- **eval_metric** (Métrica de avaliação) = **'logloss'**

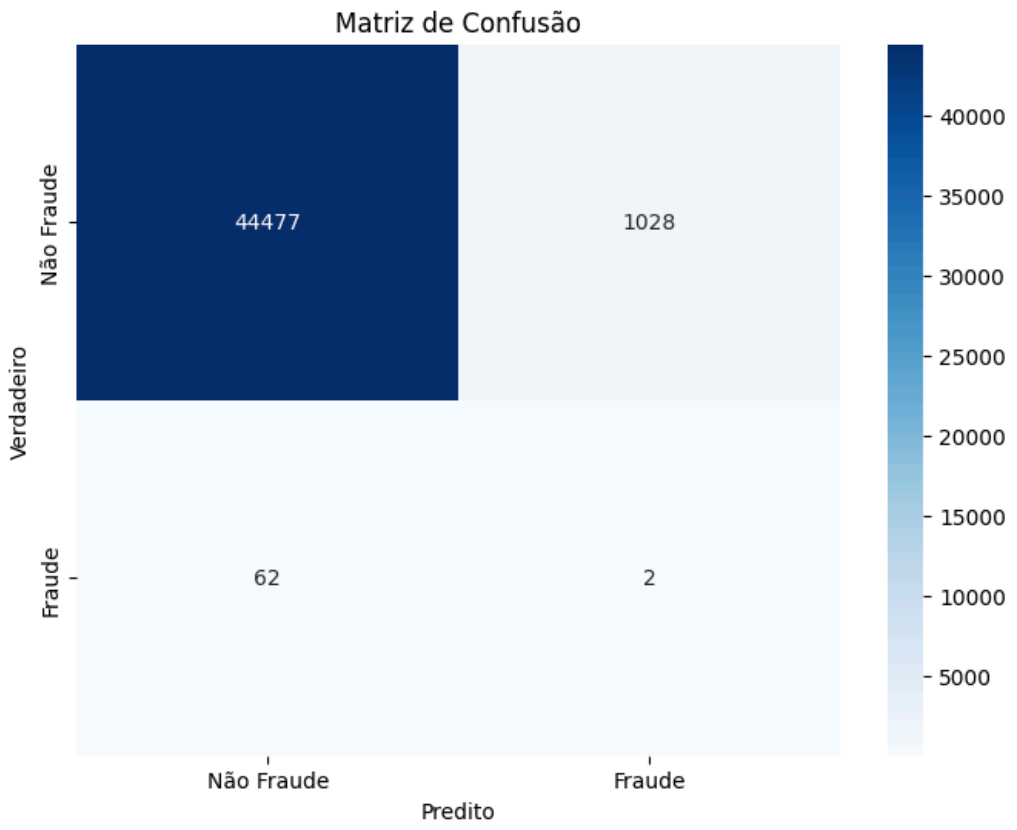
```
# Avaliar o modelo
y_pred = model.predict(X_test)
# Calcular métricas
acc = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Acurácia no teste: {acc:.4f}")
print(f"Precisão: {precision:.4f}")
print(f"Recall (Sensibilidade): {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
```

```
Acurácia no teste: 0.9761
Precisão: 0.0019
Recall (Sensibilidade): 0.0312
F1-Score: 0.0037
```

Resultado obtido do modelo

Em relação aos resultados obtidos o modelo alcançou uma **acurácia de 97.61%** no conjunto de teste logo o modelo acerta a classe majoritária ("Não Fraude") na maioria dos casos, mas isso é enganoso devido ao desbalanceamento dos dados.Obteve uma **precisão de 0.19%** o que indica que apenas 0.19% das transações classificadas como "Fraude" são realmente fraudes. Isso significa um alto número de falsos positivos (FP) — muitas transações legítimas estão sendo erroneamente bloqueadas.Um **recall de 3.12%** ou seja modelo detecta apenas 3.12% das fraudes reais. Isso é crítico, pois 96.88% das fraudes passam despercebidas (falsos negativos).E por fim um **F1-score de 0.37%** demonstrando uma combinação ruim de **precisão** e **recall**, confirmando que o modelo é ineficaz para detectar fraudes.



Matriz de confusão dos resultados obtidos

A matriz de confusão mostrada confirma o que foi afirmado anteriormente podendo se observar:

- **Verdadeiros Negativos**: 44.477 transações foram corretamente classificadas como "Não Fraude".
- **Falsos Positivos :** 1.028 transações legítimas foram erroneamente marcadas como "Fraude" (alarmes falsos).
- **Falsos Negativos (FN):** 62 fraudes reais foram classificadas como "Não Fraude" (erro grave, pois passaram despercebidas).
- **Verdadeiros Positivos (TP):** Apenas 2 fraudes foram detectadas corretamente.

De modo geral conclui-se que em um primeiro teste sem realizar técnicas de balanceamento de classes e ajustes de parâmetros do modelo observamos que esse modelo seria quase inútil para identificar fraudes, priorizando erroneamente a classe majoritária **("Não Fraude").**Acreditamos que utilizar o balanceamento de classes juntamente com RandomizedSearch para o ajuste dos parâmetros pode vir a ser benéfico.

Random Forest

O modelo Random Forest (Floresta Aleatória) é um algoritmo de aprendizado supervisionado do tipo ensemble que constrói múltiplas árvores de decisão durante o treinamento e produz a classe que é o modo das classes (classificação) ou a predição média (regressão) das árvores individuais. Ele é conhecido por sua robustez, capacidade de lidar com dados complexos e por mitigar o overfitting comum em árvores de decisão únicas.

Neste teste, o modelo foi implementado utilizando a biblioteca scikit-learn. Para o treinamento com os parâmetros fixos, foram definidos os seguintes:

- `n_estimators`: 253
- `criterion`: 'entropy'
- `max_depth`: 10
- `min_samples_leaf`: 2
- `min_samples_split`: 2
- `random state`: 42 (para reprodutibilidade)
- `class_weight`: 'balanced' (para lidar com o desbalanceamento)

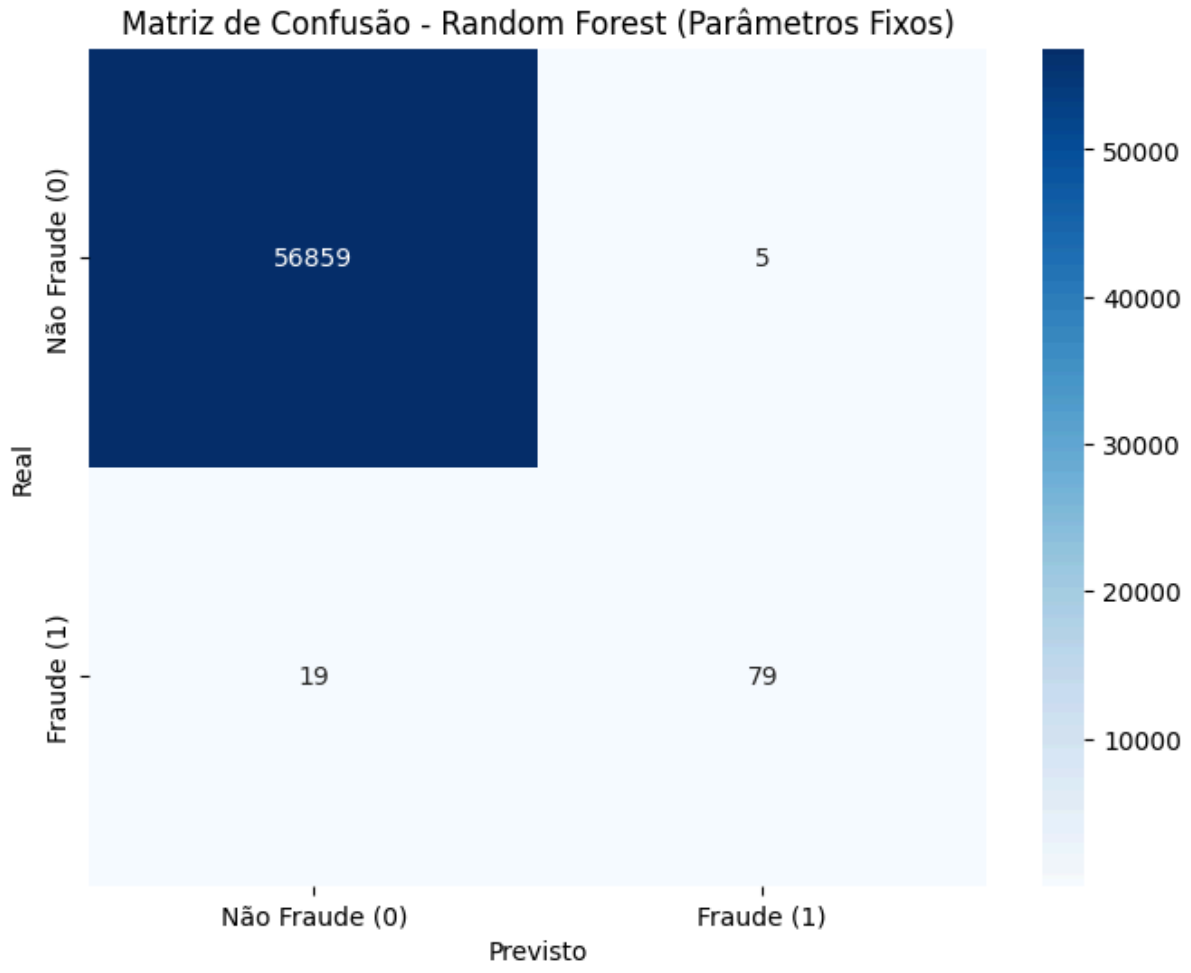
O modelo foi treinado utilizando o conjunto de dados já pré-processado (`train.csv`) e validado no conjunto de teste (`teste.csv`) com a métrica F1-score. O tempo de treinamento para esta configuração específica não foi cronometrado com precisão, mas modelos Random Forest com este número de estimadores e profundidade geralmente treinam em um tempo moderado. Após a validação, os resultados obtidos foram bastante positivos, com um F1-score macro de 0.93. Especificamente, a classe 0 (transações não fraudulentas) apresentou um F1-score de 1.00, enquanto a classe 1 (transações fraudulentas) obteve um F1-score de 0.87.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.94	0.81	0.87	98
accuracy			1.00	56962
macro avg	0.97	0.90	0.93	56962
weighted avg	1.00	1.00	1.00	56962

Fonte: Script Python `RandomForestCreditCard.ipynb` (baseado nos parâmetros e dados fornecidos)

Analisando a Matriz de Confusão, tivemos:

- **Verdadeiros Negativos:** 56.859 transações foram corretamente classificadas como "Não Fraude".
- **Falsos Positivos:** 5 transações legítimas foram erroneamente marcadas como "Fraude" (alarmes falsos).
- **Falsos Negativos (FN):** 19 fraudes reais foram classificadas como "Não Fraude" (erro grave, pois passaram despercebidas).
- **Verdadeiros Positivos (TP):** 79 fraudes foram detectadas corretamente.



Fonte: Imagem da Matriz de Confusão fornecida (gerada pelo Notebook do Random Forest)

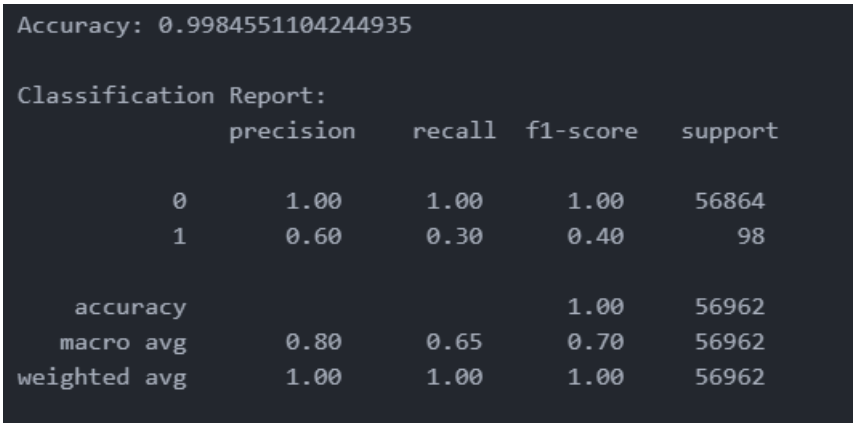
SVM

O modelo Support Vector Machine (SVM), implementado com a biblioteca scikit-learn, é um algoritmo de aprendizado supervisionado que busca encontrar um hiperplano ótimo para separar as classes no espaço de features. Esse hiperplano é escolhido de forma a maximizar a margem, que é a distância entre o hiperplano e os pontos de dados mais próximos de cada classe, conhecidos como vetores de suporte. São esses vetores de suporte que "suportam" ou definem o hiperplano e, portanto, são os elementos mais críticos do conjunto de dados para a construção do classificador.

O treinamento foi feito sem que muitos parâmetros fossem utilizado, sendo os únicos definidos:

- **kernel**='linear'
- **random_state**=42

Após aproximadamente 9 minutos de treinamento o resultado obtido foi o seguinte:



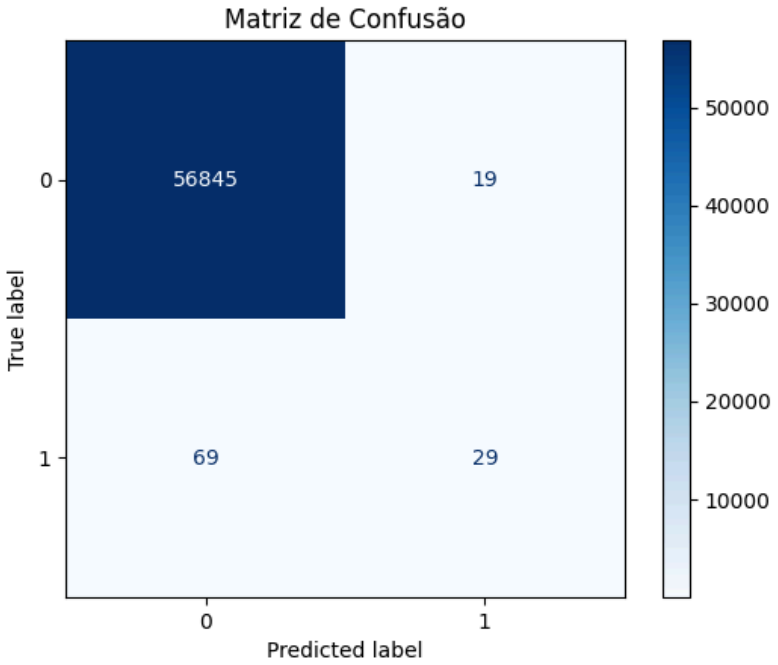
```
Accuracy: 0.9984551104244935

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.60      0.30      0.40        98

    accuracy                           1.00     56962
   macro avg       0.80      0.65      0.70     56962
weighted avg       1.00      1.00      1.00     56962
```

Fonte: Notebook svmFrauds



Fonte: Notebook svmFrauds

O modelo SVM com kernel linear conseguiu classificar os dados, apresentando uma acurácia de 99.85%. Apesar do excelente desempenho na classe majoritária (não fraudulenta), com precisão, recall e F1-score perfeitos, a alta acurácia é enganosa, pois é dominada pela classe majoritária. É notável que o modelo teve dificuldades em identificar a classe minoritária (fraudulenta). Apenas 30% das fraudes foram corretamente identificadas (recall), o que reflete o impacto do desbalanceamento dos dados no desempenho do modelo.

CatBoost

O modelo CatBoost, Categorical Boosting, é um algoritmo de aprendizado de máquina que utiliza *gradient boosting* em árvores de decisão. Ele é utilizado quando há a categorização de dados e é recomendado para a identificação de fraudes.

Para fazer a seleção dos hiperparâmetros que geraram os melhores resultados, foi utilizado o RandomizedSearch. Dadas algumas opções de parâmetros, o RandomizedSearch testou o modelo com 5 combinações aleatórias dos parâmetros e selecionou a melhor combinação. Após a avaliação, a melhor combinação de parâmetros testada foi:

- Bagging Temperature (Bootstrap Bayesiano): 9
- Border Count (número de divisões): 229
- Depth (profundidade): 6
- Iterations (número de árvores): 656
- L2 Regularization Term (termo de regularização): 5
- Learning Rate (taxa de aprendizado): 0.1451497755908629

- Random Strength (aleatoriedade das divisões): 0.013264961159866528

Após rodar os experimentos, obtivemos como F1-score macro 0.94. Sendo o F1-score da classe 0 (transações não fraudulentas) 1.00 e o F1-score da classe 1 (transações fraudulentas) 0.89. Acreditamos que a diferença entre os F1-scores se dá pelo desbalanceamento do nosso dataset, onde a classe 0 aparece diversas vezes. Abaixo é possível ver o relatório do modelo treinado:

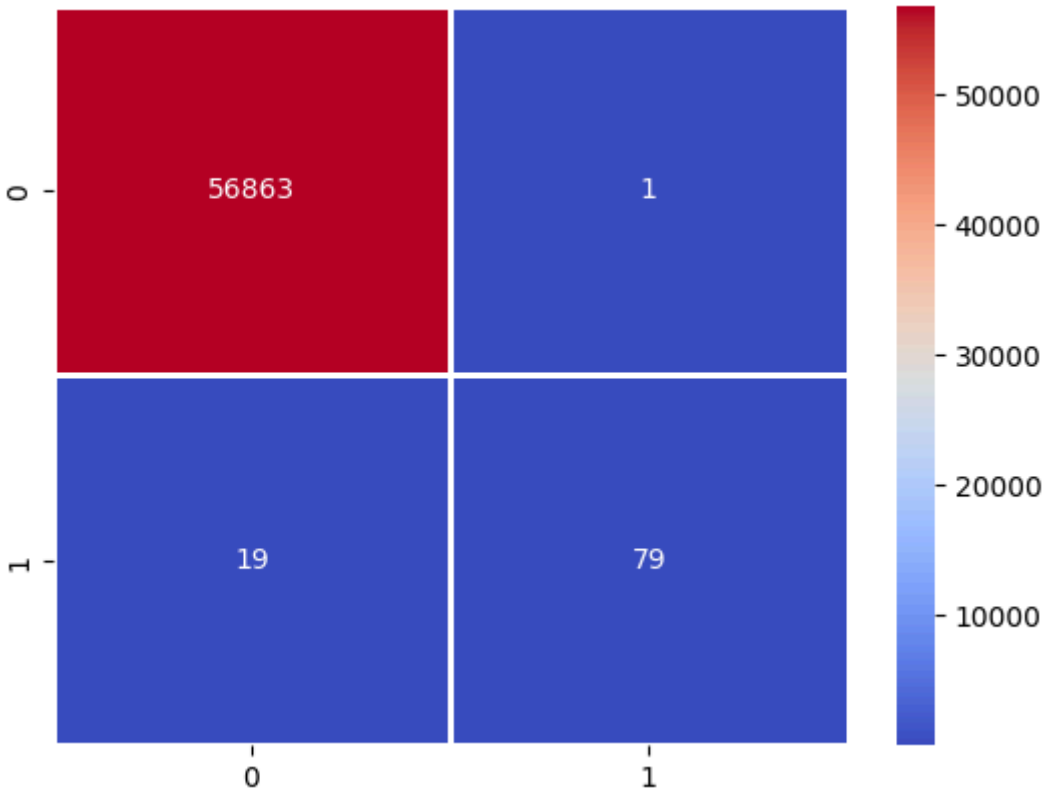
Relatório de Classificação:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	1.00	0.04	0.08	98
accuracy			1.00	56962
macro avg	1.00	0.52	0.54	56962
weighted avg	1.00	1.00	1.00	56962

Fonte: Notebook CatBoostCreditCard

Analisando a Matriz de Confusão, tivemos:

- **Verdadeiros Negativos**: 56.863 transações foram corretamente classificadas como "Não Fraude".
- **Falsos Positivos :** 1 transações legítimas foram erroneamente marcadas como "Fraude" (alarmes falsos).
- **Falsos Negativos (FN):** 19 fraudes reais foram classificadas como "Não Fraude" (erro grave, pois passaram despercebidas).
- **Verdadeiros Positivos (TP):** 79 fraudes foram detectadas corretamente.



Fonte: Notebook CatBoostCreditCard

AdaBoost

O modelo AdaBoost (Adaptive Boosting) é um algoritmo de aprendizado supervisionado que combina vários classificadores fracos — geralmente árvores de decisão rasas (stumps) — para formar um classificador forte. Ele funciona ajustando os pesos das amostras de treinamento de forma adaptativa, dando maior importância às observações que foram incorretamente classificadas em iterações anteriores.

Neste teste, o modelo foi implementado utilizando a biblioteca scikit-learn. Para o treinamento inicial, foram definidos os seguintes parâmetros:

- n_estimators: 100
- learning_rate: 0.8
- random_state: 2018

O modelo foi treinado utilizando o conjunto de dados já pré-processado e validado com a métrica F1-score. O tempo de treinamento foi de aproximadamente 1 minuto e 48 segundos. Após a validação, os resultados obtidos foram satisfatórios,

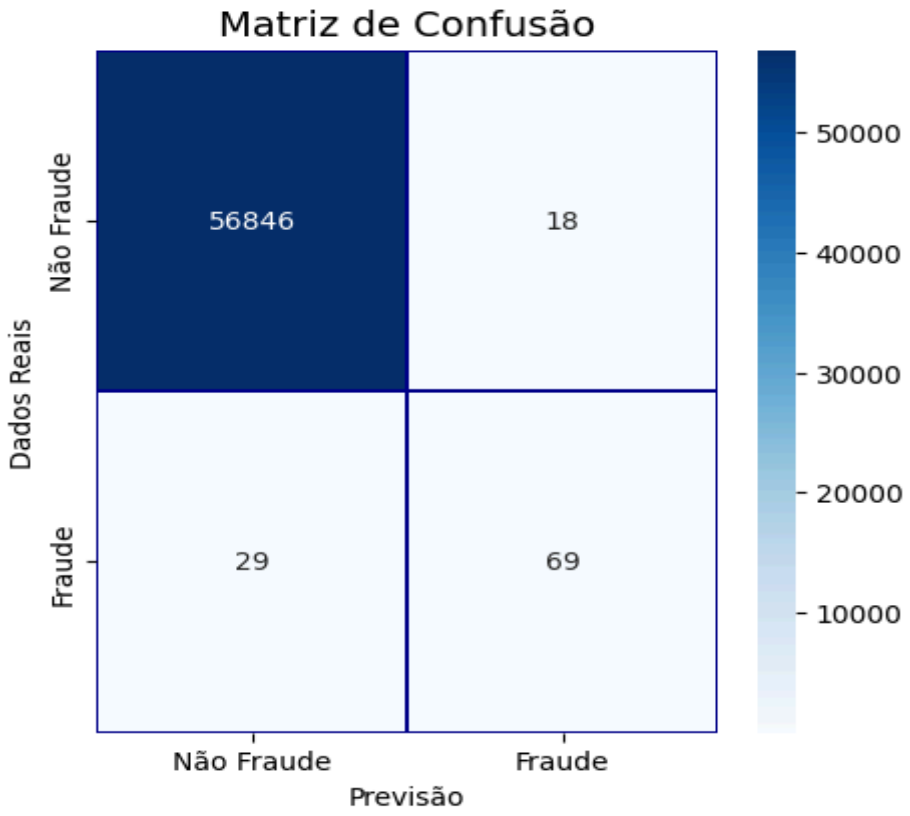
com F1-score de 0.87 Especificamente, a classe 0 (transações não fraudulentas) apresentou um F1-score de 1 enquanto a classe 1 (transações fraudulentas) obteve um F1-score de 0.75.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.79	0.70	0.75	98
accuracy			1.00	56962
macro avg	0.90	0.85	0.87	56962
weighted avg	1.00	1.00	1.00	56962

Fonte: Notebook AdaBoostCreditCard

Analisando a Matriz de Confusão, tivemos:

- **Verdadeiros Negativos**: 56.846 transações foram corretamente classificadas como "Não Fraude".
- **Falsos Positivos :** 18 transações legítimas foram erroneamente marcadas como "Fraude" (alarmes falsos).
- **Falsos Negativos (FN):** 29 fraudes reais foram classificadas como "Não Fraude" (erro grave, pois passaram despercebidas).
- **Verdadeiros Positivos (TP):** 69 fraudes foram detectadas corretamente.



Fonte: Notebook AdaBoostCreditCard

Fontes:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
https://catboost.ai/docs/en/
https://medium.com/@pwrxndr/catboost-classifier-a-simple-guide-for-everyone-48a2e3897251

Modelos Escolhidos

- **Random Forest:** Apresentou excelente desempenho na detecção de fraudes (Classe 1), com Recall de 81% e F1-score de 0.87. Utilizou balanceamento de classes (`class_weight='balanced'`).
- **CatBoost:** Após otimização com RandomizedSearch, também demonstrou forte desempenho, identificando 79 fraudes (TP) com apenas 19 Falsos Negativos (FN) e 1 Falso Positivo (FP), resultando em Recall de ~81% e F1-score de ~0.89 para a classe fraude.

Conclusão

Os modelos **Random Forest** (com balanceamento de classe) e **CatBoost** (otimizado via RandomizedSearch) foram os mais eficazes na detecção de fraudes, alcançando os maiores valores de Recall e F1-Score para a classe minoritária. Modelos como KNN, SVM (linear) e XGBoost (sem otimização/balanceamento) mostraram-se inadequados devido à baixa capacidade de identificar as transações fraudulentas reais (baixo Recall).