Money Saver Preditor

Leonardo Peres Dias, José Ronaldo Silva Henrique Filho, Jean Carlo Simpliciano do Amaral
Instituto Tecnológico de Aeronáutica
São José dos Campos, SP, Brasil
Email: {leonardo.dias.8730@ga.ita.br

***Resumo*—O projeto aqui apresentado tem como objetivo a criação de um software de ciência de dados que auxilia a previsão de fraudes realizadas em transações financeiras. Para isso, um modelo de aprendizado de máquina foi treinado em uma base de dados e seu deploy foi realizado em um website escrito em Flask. O dataset está disponível em https://www.kaggle.com/datasets/kartik2112/fraud-detection. E o repositório do projeto está disponível em https://github.com/leopers/Money-Saver-Preditor.**

***Index Terms*—Fraude, Aprendizado de Máquina, Flask, PostgreSQL, Railway**

I. Introdução

O projeto aqui apresentado tem como objetivo a criação de um *software* de ciência de dados que auxilia a previsão de fraudes realizadas em transações financeiras. Para isso, um modelo de aprendizado de máquina foi treinado em uma base de dados e seu *deploy* foi realizado em um *website* escrito em *Flask*. O *dataset* está disponível em https://www.kaggle.com/datasets/kartik2112/fraud-detection. E o repositório do projeto está disponível em https://github.com/leopers/Money-Saver-Preditor.

II. Sobre o Dataset

O dataset é um conjunto de dados simulado de transações com cartão de crédito contendo transações legítimas e fraudulentas do período de 1º de janeiro de 2019 a 31 de dezembro de 2020. Ele abrange cartões de crédito de 1000 clientes realizando transações com um grupo de 800 comerciantes.

Fonte da Simulação: Este conjunto de dados foi gerado usando a ferramenta Sparkov Data Generation | Github criada por Brandon Harris. Esta simulação foi executada durante o período de 1º de janeiro de 2019 a 31 de dezembro de 2020. Os arquivos foram combinados e convertidos em um formato padrão.

A base de dados é composta por duas tabelas: a primeira delas com 1.296.676 linhas foi utilizada como fonte de dados para o treino e posterior validação do modelo, a segunda, totalizando 55.720 linhas, foi usada como fonte de novos dados (simulação de uma situação real em que novos dados são imputados) e para o teste do modelo que foi colocado em *deploy*. A variável target é categórica e binária que indica: 0 se a transação não é fraudulenta e 1 se a transação é fraudulenta.

As features utilizadas para o treinamento do modelo foram:

amount(usd): O valor da transação em dólares americanos (USD).
lat: Latitude do local onde a transação foi realizada.
long: Longitude do local onde a transação foi realizada.
merch_lat: Latitude do local do comerciante onde a transação foi realizada.
merch_long: Longitude do local do comerciante onde a transação foi realizada.
age: Idade do titular do cartão de crédito.
merchant: Nome ou identificação do comerciante onde a transação foi realizada.
job: Ocupação ou profissão do titular do cartão de crédito.
hour_of_day: Hora do dia em que a transação foi realizada.
month: Mês em que a transação foi realizada.
day_of_week: Dia da semana em que a transação foi realizada.

III. *Deployment* da base de dados

A base de dados foi projetada em quatro tabelas na tentativa de obtenção de uma *database tidy* e 3NF, conforme mostra a figura a seguir.

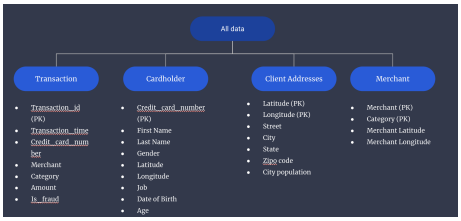

Figura 1: Estruturação da base de dados

Assim, no processo de *deployment*, o PostgreSQL foi escolhido como o sistema de gerenciamento de banco de dados devido à sua robustez e conformidade com o padrão relacional do SQL. A *database* foi hospedada no site Railway, que oferece excelentes soluções em nuvem.

IV. Treinamento e validação do modelo

A. *Preprocessing Pipeline*

Como as operações de pré-processamento dos dados devem ser declaradas previamente e posteriormente fitadas, ao invés de serem declaradas *ad-hoc*, a classe *PreprocPipeline* foi declarada, de modo a modularizar as operações:

- Seleção das *features*.
- Uso de Standard Scaler nas variáveis numéricas.
- Uso de One-Hot-Encoding nas variáveis categóricas.
- *Save* e *load* do *pipeline*.

B. Model Pipeline

Para o treinamento, foi criada a classe *ModelPipeline* que encapsula as seguintes operações:

- Pré-processamento de dados (classe *PreprocPipeline*).
- Random Undersampling e SMOTE para rebalanceamento do dataset, que é fortemente desbalanceado.
- Fit, predict e score para o modelo.
- *Save* e *load* do *modelo*.

C. Validação

Para a validação do modelo, foram propostos 3 modelos distintos.

```
models = [
(  Logistic   Regression ,
   LogisticRegression(max_iter=10000,
   random_state=42)),
(  Decision   Tree ,
   DecisionTreeClassifier(random_state
   =42, max_depth=10)),
(  Random   Forest ,
   RandomForestClassifier(random_state
   =42, max_depth=10, n_estimators
   =100)),
]
```

A partir daí, foi utilizada uma validação cruzada estratificada de 10 folds, garantindo pareamento entre os folds de cada modelo e manutenção da distribuição das classes entre os folds.

Visto que queremos penalizar principalmente a ocorrência de falsos negativos, porém sem perder controle sobre os falsos positivos deliberadamente, foram analisadas principalmente as métricas de recall e especificidade. Obtemos, portanto, as seguintes métricas de validação:

Modelo	Recall	Especificidade	Acurácia
Logistic Regression	0.86	0.90	0.90
Decision Tree	0.88	0.96	0.96
Random Forest	0.76	0.93	0.93

Tabela I: Desempenho dos modelos propostos (Média)

Modelo	Recall	Especificidade	Acurácia
Logistic Regression	0.010	0.002	0.001
Decision Tree	0.012	0.003	0.003
Random Forest	0.018	0.011	0.011

Tabela II: Desempenho dos modelos propostos (Desvio Padrão)

Visto isso, o modelo escolhido foi a *Decision Tree*.

D. Deployment e WebService

Para o *deploy*, o modelo foi treinado em todo o dataset e salvo no formato .pkl. Esse treinamento foi realizado com pesos das classes balanceados (*Balanced class weights*) para uma melhor performance em datasets altamente desbalanceados como o nosso.

Para o frontend da página foram utilizados HTML e CSS3, sendo o Flask empregado na renderização.

- Estrutura básica dos inputs e formulários colocados no HTML.
- CSS3 empregado na estilização.
- O Flask renderiza templates HTML e gerencia rotas e requisições HTTP.

Já o backend foi realizado a partir das seguintes metodologias:

- Os dados são submetidos em forma de request direcionados pelo arquivo `routes.py` para cumprir suas especificidades.
- Inserção de dados na database: Um cursor apontando para a base de dados pega os dados inseridos e faz uma inserção no banco de dados.
- Check Fraud: Os dados das features obtidas no treinamento são submetidos e passam pelo modelo de treinamento para predição.
- Feedback: Depois do resultado da predição, o usuário pode inserir se a predição foi correta ou não.