Identificação de Fraudes em Transações Financeiras

Beatriz Francisco (118638) e Mariana Marques (118971)

1. Introdução

No âmbito da cadeira de Métodos Probabilísticos para Engenharia Informática propusemo-nos a desenvolver um trabalho prático que avalia se uma dada transação financeira pode ser considerada fraudulenta ou legítima. Para tal, aplicamos conceitos abordados ao longo da unidade curricular de forma a conseguirmos de diferentes formas concluir sobre a transação a analisar. Nomeadamente, o classificador de Naive Bayes, o Bloom Filter e o MinHash. Para o desenvolvimento do projeto, dispusemos de um *dataset*[1] que modificámos para se enquadrar às nossas necessidades. Este conjunto de dados serviu tanto para treinar os modelos como para testar a sua eficácia e, em última instância, possibilitar uma aplicação conjunta das técnicas. Desta forma, o trabalho procura demonstrar como cada método utiliza uma perspetiva distinta ao problema da detecção de fraudes e de que forma a sua combinação pode aumentar a robustez e a eficácia do sistema final.

2. Classificador Naive Bayes

O objetivo deste módulo é avaliar se uma transação pode ser considerada fraudulenta, tomando como base um conjunto de variáveis que descrevem as transações (como o valor, o *merchant*, a localização, hora, data, categoria da compra e informações sobre o titular do cartão). Para o correr é só clicar no botão *Run* como normalmente, na *main.m* na pasta do Naive Bayes, que irá mostrar as métricas e matrizes confusão automaticamente uma vez que utiliza o *dataset* como testes. As últimas são geradas para identificar os erros de classificação (falsos positivos e falsos negativos) e os acertos (verdadeiros positivos e verdadeiros negativos).

O *dataset* usado foi dividido num conjunto de dados para treino e outro para teste, usando *cvpartition()* que vai alterar os números das matrizes de confusão apresentadas, uma vez que não divide sempre o *dataset* da mesma maneira, ou seja, o número de transações fraudulentas varia sempre. Adicionalmente, métricas como precisão, recall e F1-score foram calculadas para entender as capacidades do modelo para balancear a detecção de fraudes reais e minimizar falsas fraudes. Utilizamos, também várias percentagens para ver como os valores iriam ser afetados com diferentes quantidades de transações para treino e concluímos que apesar de o aumento na percentagem de treino não alterar significativamente a exatidão (mantendo-se entre 0.9958 e 0.9972), as métricas relacionadas à detecção de fraudes (precisão, recall e F1-score) foram mais sensíveis. Os valores indicam que, embora a exatidão global se mantenha, uma maior quantidade de dados de treino contribui para uma identificação mais eficaz de fraudes.

Treino (%)	Teste (%)	Exatidão	Precisão	Recall	F1-score
60	40	0.9966	0.5000	0.2059	0.2917
50	50	0.9958	0.3333	0.1818	0.2353
70	30	0.9967	0.6429	0.3103	0.4186
90	10	0.9972	0.5000	0.2857	0.3636

Tabela 01-Métricas Naive Bayes consoante percentagem de treino

[1] *Credit Card Transactions Fraud Detection Dataset*

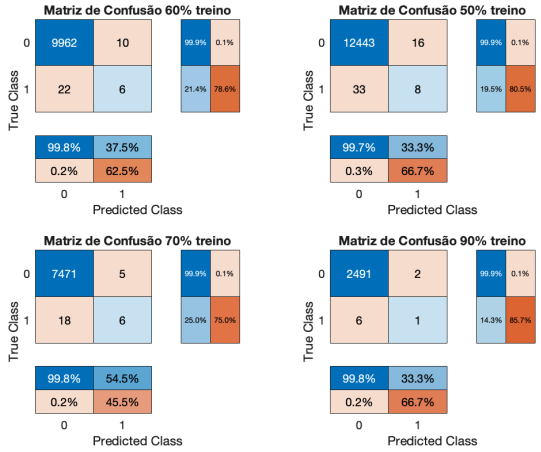

Figura 01 - Exemplo de Matriz Confusão

Os baixos valores de recall, devem-se a um desbalanceamento de fraudes baixas no *dataset*.

Em título de curiosidade, também foi acrescentado as métricas baseadas só nas categorias das transações para ver como iriam variar.

3. Bloom Filter

O módulo do Bloom Filter foi desenvolvido com a finalidade de verificar se uma transação já foi sinalizada anteriormente como fraudulenta. Na nossa aplicação decidimos só utilizar as colunas relativas ao número do cartão, ao merchant e a categoria da compra, uma vez que achámos que eram as colunas com mais relevância a esta aplicação. Ou seja, escolhemos uma transação aleatória do *dataset* e comparamos às transações fraudulentas que guardamos no Bloom Filter para assim verificar se é alguma delas e já está potencialmente assinalada como fraudulenta. Irá servir futuramente na aplicação conjunta como um primeiro filtro rápido de verificação se a transação é fraudulenta ou não, poupando tempo se já estiver sido marcada. Pode no entanto, ter algumas limitações sendo uma delas a possível ocorrência de falsas fraudes. Poderíamos ainda ter colocado uma maneira de atualizar o Bloom filter regularmente com novas fraudes, mas não achamos que fosse relevante devido ao contexto do projeto, pois qualquer transação adicionada seria inserida na aplicação conjunta.

Para correr o programa e testá-lo basta correr a *main.m* dentro da pasta do Bloom Filter.

4. MinHash

O objetivo do módulo do MinHash é identificar similaridades entre transações para detetar padrões que identifiquem as transações como fraudes. Essa análise, no nosso código, é baseada através das informações relativas ao número do cartão, o merchant, a categoria da compra e o seu valor. Após ter acesso a todas essas informações, procedemos à criação de um conjunto de shingles, onde cada um tem informações relativas a cada transação.

À semelhança do Bloom Filter, recorremos a funções hash de modo a se conseguir calcular o valor das assinaturas através da função *generateSignatures()*. Assim que todas as assinaturas estejam calculadas, é possível calcular a matriz de similaridade das transações, uma vez que com a função *jaccardSimilarity()* é aplicada a fórmula relativa à similaridade de Jaccard, que envolve os valores das assinaturas.

De forma a facilitar a identificação de padrões entre as transações, decidimos agrupar transações semelhantes em conjuntos quando a sua similaridade fosse superior a 80%. Para tal, criámos a função *detectFraudClusters()* cujo objetivo é identificar transações que sejam similares e agrupá-las no mesmo

conjunto. Quando já tiverem sido identificados todos os conjuntos de fraudes semelhantes, a matriz de similaridade é reorganizada com esses mesmos conjuntos.

Para tornar mais perceptível o que este módulo faz, construímos alguns gráficos que ajudam a demonstrar o que pretendíamos que fosse realizado. No entanto, como os coeficientes utilizados nas funções hash são calculados de forma aleatória, os gráficos não são sempre iguais, uma vez que os valores das assinaturas também irão mudar e, consequentemente, a similaridade entre as transações.

Estes gráficos são visíveis após a execução do ficheiro *main.m* que se encontra na pasta relativa ao minHash, não sendo preciso nenhuma especificidade extra para correr o programa.

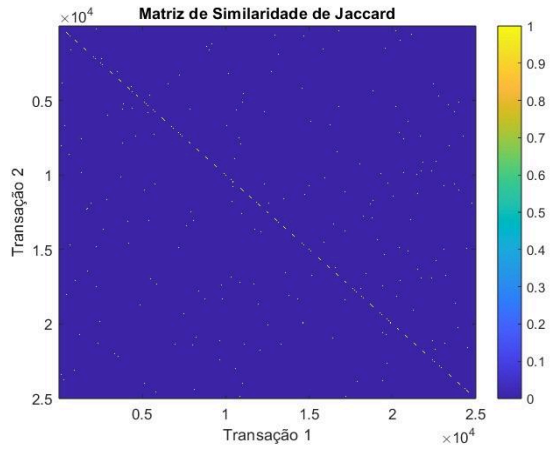

Figura 02 - Matriz de Similaridade
(Antes de estar organizada)

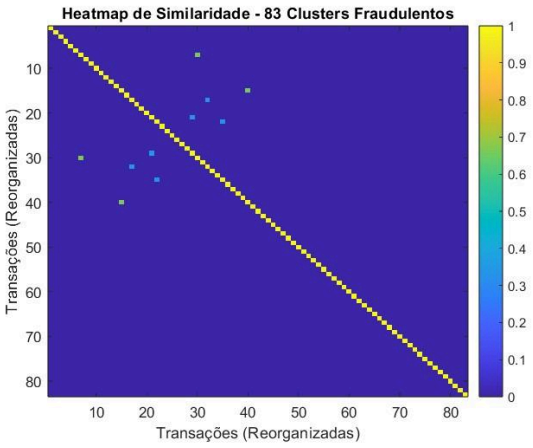

Figura 03 - Matriz de Similaridade (Fraudes)
(Depois de estar organizada)

4.1. Função *generateSignatures()*

Esta função necessita como argumentos o conjunto de shingles, isto é, as transações, o número de funções hash que vão ser utilizadas e as próprias funções que se vão querer utilizar.

Quando todas as assinaturas estiverem calculadas, a função retorna esses mesmos valores que serão úteis para a continuação da execução do programa.

4.2. Função *jaccardSimilarity()*

O objetivo desta função é simplesmente calcular a similaridade entre as transações recorrendo à fórmula da similaridade de Jaccard, que envolve a interseção e união entre as próprias transações.

4.3. Função *detectFraudClusters()*

Por fim, desenvolvemos esta função, que através da similaridade entre as transações fraudulentas, permite agrupar em conjuntos as transações de modo a identificar padrões que permitam concluir sobre a natureza da transação.

5. Aplicação Conjunta

Ao longo da execução de cada um dos módulos pedidos, as variáveis mais cruciais foram guardadas no ficheiro *dados.mat* para que fosse possível realizar uma aplicação conjunta de todos os módulos desenvolvidos. Assim, para o correto funcionamento desta etapa do projeto, é necessário que, anteriormente, tenham sido executados individualmente os módulos do Naive Bayes, Bloom Filter e MinHash.

O intuito desta aplicação conjunta é pedir ao utilizador para definir os parâmetros de uma transação à sua escolha e, através desses dados, irá saber se a sua transação pode ser considerada fraudulenta.

A primeira funcionalidade a ser utilizada é o Bloom Filter, o que leva a que as primeiras informações sobre a transação a serem pedidas sejam o número do cartão, o merchant e a categoria da compra, uma vez que o Bloom Filter que desenvolvemos baseia-se nesses dados. De seguida, aplicamos o filtro à transação em questão e caso esteja presente no filtro, o programa termina imediatamente identificando a transação como fraudulenta.

Nos casos em que a transação passa essa verificação, são pedidos os restantes detalhes da transação, recorrendo à função *detailsTransaction()* de modo a que se possa prever a legitimidade da compra através do classificador Naive Bayes. De seguida, acontece o mesmo procedimento de antes, ou seja, caso o classificador identifique a transação como fraude, surgirá uma mensagem que contém essa informação.

Se a transação continuar a ser considerada como segura, recorremos ainda ao MinHash para verificar se existe alguma fraude semelhante à transação que está a ser analisada para, por fim, ser possível concluir sobre a natureza da compra efetuada.

Assim, para que se consiga testar o programa, o utilizador apenas tem de colocar, à vez, os parâmetros da transação que quer verificar se é fraude ou não.

5.1. Testes

Ao utilizar uma transação da base de dados que tenha sido considerada falsa, conseguimos demonstrar que o programa termina imediatamente, uma vez que já reconhece essa transação. No entanto, se ele não identificar a transação como um elemento do Bloom Filter, vai pedir mais informações sobre a compra, como foi detalhado anteriormente.

```
>> apConjunta
Introduz uma transação para avaliar se é fraude ou não
Credit Card Number: 3560725013359370
Merchant: fraud_Hamill-D'Amore
Category: health_fitness
Essa transação é considerada fraudulenta segundo o Bloom Filter.
```

Figura 04 - Demonstração Bloom Filter

```
>> apConjunta
Introduz uma transação para avaliar se é fraude ou não
Credit Card Number: 2291163933867240
Merchant: fraud_Haley Group
Category: personal_care
Date and time of the transfer (yyyy-MM-dd HH:mm:ss): 2020-06-21 12:17:21
Amount: 25.63
First Name: Jeff
Last Name: Smith
Gender (M/F): M
Street: 9345 Spencer Junctions Suite 183
City: Colorado
State: CO
ZIP: 80951
Latitude: 38,8881
Longitude: -104,6556
City Pop: 525713
Job: Professor
DOB: 1984-09-14
Transaction Number: 8be473af4f05fc6146ea55ace73e7ca2
Unix Time: 1371816971
Merchant Latitude: 33,195225
Merchant Longitude: -97,919284
Essa transação não é fraudulenta
```

Figura 05 - Funcionamento do programa

6. Funcionalidades Extra

Além da aplicação conjunta, criámos ainda um programa onde os três módulos também são executados simultaneamente mas é permitido ao utilizador escolher quais são as categorias de dados que quer utilizar no classificador de Naive Bayes, para apenas prever a legitimidade da transação com base nesses dados.

Outra funcionalidade deste programa é o facto de termos adicionado um gráfico que permite analisar o número de fraudes que existem em cada hora do dia e, consequentemente, a hora mais provável de as transações serem fraudulentas.

```
>> mainGlobal
Categorias disponíveis para o Naive Bayes:
1. cc_num
2. merchant
3. category
4. amt
5. first
6. last
7. gender
8. street
9. city
10. state
11. zip
12. lat
13. long
14. city_pop
15. job
16. dob
17. trans_num
18. unix_time
19. merch_lat
20. merch_long
21. Todas as Opções
Selecione as categorias desejadas digitando os números separados por vírgulas.
Exemplo: [1, 3, 5]: [2, 6, 10]
Exatidão (Accuracy): 0.9968
Precisão: 0.0000
Recall: 0.0000
F1-Score: 0.0000
```

Figura 06 - Funcionamento do programa

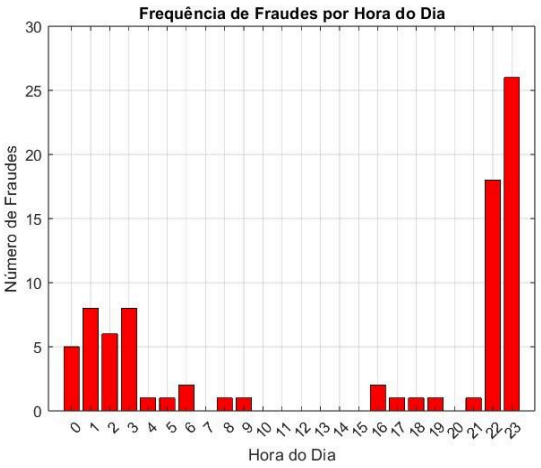

Figura 07 - Número de Fraudes por Hora

7. Conclusão

A integração dos três módulos – Naive Bayes, Bloom Filter e MinHash – demonstrou ser uma abordagem mais completa e eficaz para a detecção de fraudes. Enquanto o Bloom Filter fornece um filtro inicial rápido, ao identificar transações já previamente marcadas como fraudulentas, o Naive Bayes avalia a probabilidade de ser fraude com base nas características da transação. Finalmente, o MinHash deteta padrões de semelhança entre transações, permitindo identificar agrupamentos de fraudes semelhantes. Esta combinação potencia as vantagens de cada método, atenuando as suas limitações individuais. Embora o sistema tenha demonstrado resultados promissores, algumas limitações foram observadas: desbalanceamento do dataset (o baixo número de fraudes no conjunto de dados afetou o desempenho do Naive Bayes em métricas como recall, o que indica que mais dados fraudulentos poderiam melhorar o modelo) e atualização do Bloom Filter (apesar de eficiente, o Bloom Filter não é dinâmico, ou seja, novas fraudes não são automaticamente incorporadas, então uma atualização periódica do filtro seria ideal em contextos reais). Em suma, o sistema desenvolvido é um exemplo prático e eficaz de como métodos probabilísticos e algoritmos de hashing podem ser aplicados para resolver problemas críticos em segurança financeira, oferecendo um balanceamento ideal entre velocidade, precisão e complexidade.