Deteção de fraudes em cartões de crédito

Relatório final projeto 2

Sistemas de Gestão de Dados
2016/2017

André Pinho, <apinho@student.dei.uc.pt>
Rafael Pinho, <rfpinho@student.dei.uc.pt>

Tabela de conteúdos

Glossário

MLib - biblioteca de machine learning escalável do apache spark
PCA - transformação Principal Component Analysis
GBTs - classificador do apache spark chamado Gradient-Boosted Tree
ANN - Artificial Neural Network
BBN - Bayesian Belief Network
SVM - Support Vector Machine

Resumo

Este trabalho teve como principal objetivo a deteção de fraudes em cartões de crédito aplicando algoritmos de inteligência artificial. Com recurso ao apache spark foram treinados vários classificadores capazes de prever a classificação de transações como legítimas ou fraudulentas. Também comparámos o desempenho e a precisão de vários classificadores disponíveis na biblioteca MLib do spark na classificação das transações. Recorrendo à paralelização dos algoritmos de classificação comparou-se o ganho de desempenho (speedup) de execução numa só máquina e em cluster com recurso ao apache hadoop.

I. Introdução

O uso de cartões de crédito tem aumentado significativamente nos últimos anos e com isso o número de fraudes também aumentou. De acordo com o banco central europeu, durante 2014 foram perdidos 1,33 mil milhões de euros em fraudes na zona euro, o que representa um aumento de 14.8% em relação a 2013. Além disso, os pagamentos com meios não tradicionais (internet, móveis, etc.) representaram 60% das fraudes, enquanto que em 2010 era de 44%. Isto abre novos desafios à medida que surgem novos padrões de fraude, que fazem com que os sistemas atuais de deteção de fraude tenham menos sucesso na prevenção dessas fraudes.

O dataset utilizado neste trabalho provém de um desafio publicado no kaggle, que disponibiliza uma plataforma para competições de ciência dos dados para empresas e investigadores. [1] Este dataset contém transações feitas por cartões de crédito em setembro de 2013 por titulares de cartões europeus. Este dataset apresenta transações que ocorreram em dois dias, onde existem 492 fraudes em 284.807 transações. O dataset é altamente desequilibrado, em que a classe positiva (fraude) representa apenas 0.172% de todas as transações.

Os dados contêm apenas variáveis de entrada numéricas que são o resultado de uma transformação PCA. Infelizmente, devido a questões de confidencialidade, o dataset não fornece as features originais e outras informações gerais sobre os dados. As features V1, V2, ... V28 são os principais componentes obtidos com PCA, as únicas features que não foram transformadas com PCA são o tempo, o montante e a classe. A feature 'Time' contém os segundos decorridos entre cada transação e a primeira transação do dataset, a feature 'Amount' contém o montante da transação, e a feature 'Class' é a variável de resposta e toma valor 1 em caso de fraude e 0 caso contrário.

Neste trabalho pretende-se fazer uma análise exploratória dos dados, e implementar vários tipos de classificadores que possam ser executados em cluster para treinar modelos de classificação de modo a prever as transações fraudulentas com elevada precisão. No final pretendemos comparar o desempenho e a precisão entre os vários classificadores, e também comparar o ganho de desempenho na execução do algoritmo num único nó e em cluster.

Na análise exploratória procurámos relacionar o número de transações e o número de fraudes com as várias horas do dia, e encontrar os atributos mais significativos para a

classificação. Para esta etapa iremos utilizar a linguagem de programação Python, recorrendo às bibliotecas pandas e matplotlib para a exploração dos dados.
Na implementação do classificador escalável utilizámos o apache spark de alto desempenho, que permite paralelizar a execução dos algoritmos em cluster. O apache spark fornece bibliotecas com vários tipos de classificadores para problemas de classificação. Além disso permite integrar com o apache hadoop para que os algoritmos possam ser paralelizados e executados em vários nós. [2]
Este relatório é constituído por quatro partes, a primeira para o estado da arte, a segunda para a análise exploratória dos dados, a terceira para a implementação da versão escalável do classificador, e a última para análise de resultados e conclusões.

II. Estado da arte

Nesta secção iremos descrever os tipos de classificadores disponibilizados pela biblioteca MLib do apache spark, e descrever alguns trabalhos relacionados com o tema do trabalho.

A. Tipos de classificadores do spark

Este trabalho baseia-se na classificação de transações de cartões de crédito a partir das suas features. Para problemas de classificação o spark apresenta os seguintes classificadores que analisaremos de seguida: naive Bayes, regressão logística, árvores de decisão, florestas aleatórias e árvores com gradient-boosted. [3]

1. Naive Bayes

O naive Bayes é um classificador multiclasse probabilístico baseado no teorema de Bayes que presume que dois eventos são independentes, ou seja, que a ocorrência de um evento não afeta a probabilidade de outro evento ocorrer. O algoritmo usa os dados de treino para calcular a distribuição de probabilidade condicional de cada feature, e de seguida, aplica o teorema de Bayes.

2. Regressão logística

A regressão logística é um método popular para prever uma resposta categorizada. É um caso especial dos modelos lineares generalizados que prevê a probabilidade dos resultados. Na biblioteca spark MLib, a regressão logística pode ser usada para prever um resultado binário usando a regressão logística binomial, ou pode ser usada para prever um resultado multiclasse usando a regressão logística multinominal.

3. Árvores de decisão

As árvores de decisão são usadas para problemas de classificação e regressão. Para tarefas de classificação a árvore de decisão é o algoritmo que apresenta o melhor desempenho. A biblioteca spark MLib suporta árvores de decisão para classificação binária e multiclasse, e para regressão usando recursos contínuos e categorizados.

4. Florestas aleatórias
As florestas aleatórias são um conjunto de árvores de decisão. São um dos modelos de aprendizagem mais bem-sucedidos para classificação e regressão. Elas combinam muitas árvores de decisão a fim de reduzir o risco de overfitting. A biblioteca spark MLib suporta florestas aleatórias para classificação binária e multiclasse, e para regressão usando features contínuas e categorizadas.

5. Árvores com gradient-boosted
As árvores com gradient-boosted (GBTs) são conjuntos de árvores de decisão. As GBTs treinam iterativamente as árvores de decisão a fim de minimizar uma função de perda.
A biblioteca spark MLib suporta GBTs para classificação binária, usando recursos contínuos e categorizados. As GBTs são construídas a partir da implementação das árvores de decisão.

B. Trabalhos relacionados
Dado que este trabalho se baseia na classificação de transações de cartões de crédito a partir das suas features, durante o trabalho relacionado procurámos alguns artigos que analisaremos de seguida sobre técnicas e métodos deteção de fraudes em cartões de crédito e também sobre algumas estratégias de engenharia de features para deteção de fraudes em cartões de crédito.

1. Credit card fraud detection using bayesian and neural networks - Deteção de fraudes em cartões de crédito usando redes neuronais e bayesianas: [4]
Este artigo fez uma comparação entre duas técnicas de machine learning: redes neuronais artificiais (ANN - Artificial Neural Network) e redes de crenças bayesianas (BBN - Bayesian Belief Network).
O artigo mostrou que podem ser obtidos bons resultados na deteção de fraudes tanto com as redes neuronais artificiais como com as redes de crença bayesianas. Na comparação realizada, as redes de crença bayesianas apresentaram melhores resultados relativos à precisão na deteção de fraude e um período de treino mais curto, embora o processo de deteção de fraude seja mais rápido com as redes neuronais artificiais.

2. Analysis on credit card fraud detection methods - Análise dos métodos de deteção de fraudes em cartões de crédito: [5]
Este artigo apresenta um levantamento de várias técnicas utilizadas em mecanismos de deteção de fraude em cartões de crédito, e avalia cada metodologia com base em determinados critérios de design. Algumas das técnicas avaliadas no artigo:
- A rede neural CARDWATCH baseada em data mining proposta por Aleskerov, mostra uma boa precisão na deteção de fraude e uma velocidade de processamento elevada, embora seja limitada a uma rede por cliente.
- Os sistemas Fuzzy Darwinian de deteção de fraude melhoram a precisão do sistema, uma vez que a taxa de deteção de fraude nos sistemas de Darwin Fuzzy em termos de true positive (TPR) é de 100%, o que se traduz em bons resultados na deteção de transações fraudulentas.

- No modelo Hidden Markov a taxa de deteção de fraude (FDR) é baixa em comparação com outros métodos existentes.

3. Data mining for credit card fraud, a comparative study - Data mining para fraudes de cartões de crédito, um estudo comparativo [6]
Este artigo avalia o desempenho de duas técnicas avançadas de data mining, florestas aleatórias e support vector machines (SVM) juntamente com regressão logística, para deteção de fraudes de cartões de crédito.
As florestas aleatórias apresentaram melhor desempenho e capturaram mais casos de fraude com menos falsos positivos em árvores profundas, que é uma consideração importante no uso real de modelos de deteção de fraude. A regressão logística manteve um desempenho semelhante com diferentes níveis de subamostragem, enquanto que o desempenho das SVM tem tendência a aumentar em menores proporções de fraude nos dados de treino.

4. Feature engineering strategies for credit card fraud detection - Estratégias de engenharia de features para deteção de fraudes em cartões de crédito [7]
Este artigo mostra a importância de utilizar features que analisem o comportamento dos utilizadores de cartões de crédito quando se constrói um modelo de deteção de fraudes de cartões de crédito.
Mostra que, ao pré processarmos os dados para incluir o comportamento recente do utilizador, o desempenho aumenta mais de 200% em comparação com o uso de informações brutas da transação. Além disso, estende a abordagem atual para analisar o comportamento do utilizador, propondo um novo conjunto de features que permitem desenvolver relações complexas dos dados. De seguida, propõe um método para analisar o comportamento periódico do tempo de uma transação usando a distribuição de Von Mises. O novo conjunto de features proposto aumenta o desempenho em 252% e 287%, respetivamente.

III. Análise exploratória de dados

Nesta seção iremos fazer a análise exploratória de dados com Python, fazendo uma análise preliminar dos dados do dataset e descobrindo quais são os atributos mais significativos para a classificação das transações, com o objetivo de melhorar a precisão do classificador.

A. Explorando os dados

Como podemos ver na imagem abaixo através dos 5 primeiros registos do dataset, os dados são na sua maioria transformados a partir da sua forma original por razões de confidencialidade. Apenas os atributos Time, Amount e Class (1 - transação fraudulenta ou 0 - transação normal) são reais.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...

	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

[5 rows x 31 columns]

Pela análise das estatísticas que resumem cada um dos atributos do dataset da imagem abaixo podemos concluir que:
- O tempo varia entre 0 e 172.792 segundos (48 horas), o que significa que as transações ocorreram num intervalo de 2 dias.
- A média dos montantes transacionados é de 88.35 €, sendo o montante transacionado mais elevado de 25691,16 €.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16	-3.137024e-15	...
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...

	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
count	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.000000
mean	1.537294e-16	7.959909e-16	5.367590e-16	4.458112e-15	1.453003e-15	1.699104e-15	-3.660161e-16	-1.206049e-16	88.349619	0.001727
std	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.041527
min	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.000000
25%	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.000000
50%	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.000000
75%	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.000000
max	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.000000

[8 rows x 31 columns]

Como podemos ver na imagem abaixo os dados não apresentam valores em falta, o que torna as análises um pouco mais fácies.

```
V1        0          V17       0
V2        0          V18       0
V3        0          V19       0
V4        0          V20       0
V5        0          V21       0
V6        0          V22       0
V7        0          V23       0
V8        0          V24       0
V9        0          V25       0
V10       0          V26       0
V11       0          V27       0
V12       0          V28       0
V13       0          Amount    0
V14       0          Class     0
V15       0          dtype: int64
V16       0
```

De seguida vamos analisar de forma gráfica como se compara o atributo tempo entre transações fraudulentas e normais para perceber se existe alguma relação.

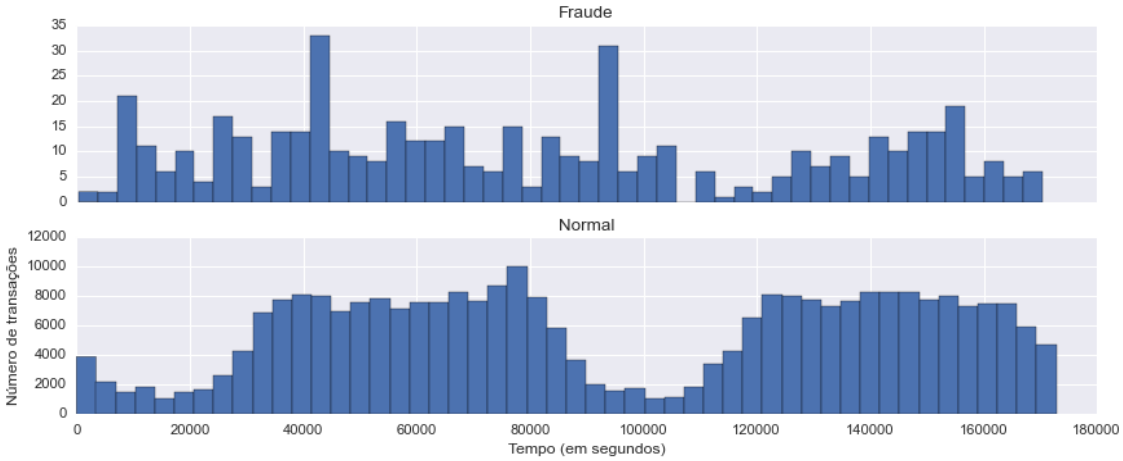


Gráfico 1 - Transações normais e fraudulentas ao longo do tempo

O atributo 'Tempo' é bastante semelhante em ambos os tipos de transações. As transações fraudulentas são distribuídas de forma mais uniforme, enquanto que as transações normais têm uma distribuição cíclica porque durante a noite o número de transações decresce.

De seguida vamos analisar se os montantes das transações diferem entre os dois tipos de transações.

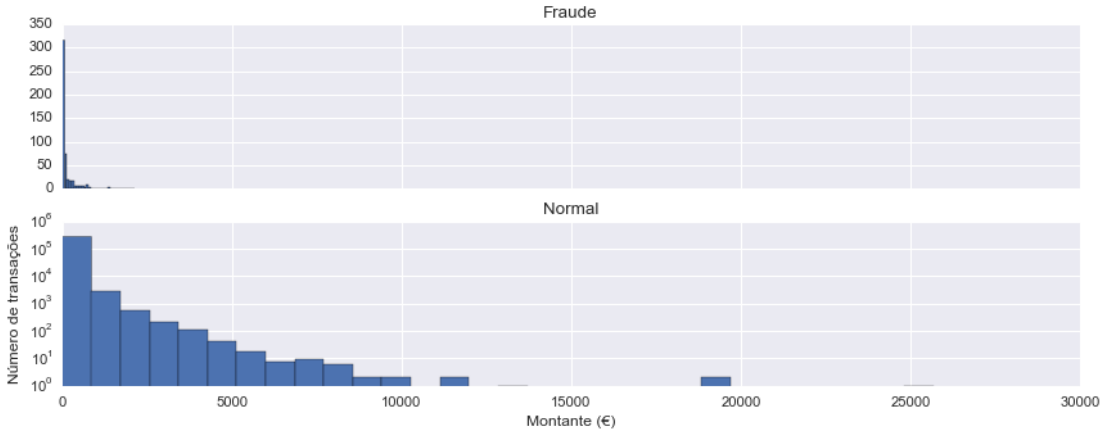


Gráfico 2 - Montantes das transações normais e fraudulentas

```
Fraude                                  Normal
count     492.000000                    count    284315.000000
mean      122.211321                    mean         88.291022
std       256.683288                    std         250.105092
min         0.000000                    min           0.000000
25%         1.000000                    25%           5.650000
50%         9.250000                    50%          22.000000
75%       105.890000                    75%          77.050000
max      2125.870000                    max       25691.160000
Name: Amount, dtype: float64            Name: Amount, dtype: float64
```

A maioria das transações são de pequenas quantias, inferiores a 100 €. As transações fraudulentas têm um valor máximo muito inferior ao das transações normais, 2125,87 € vs. 25691,16 €.

Agora vamos comparar o tempo com o montante e ver se podemos aprender algo novo.

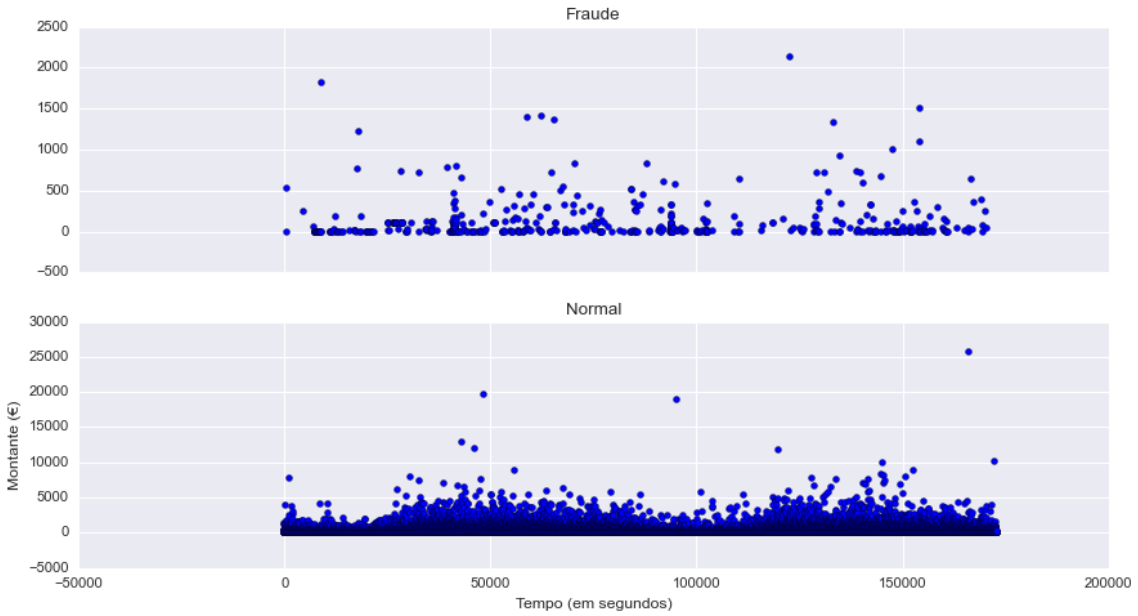
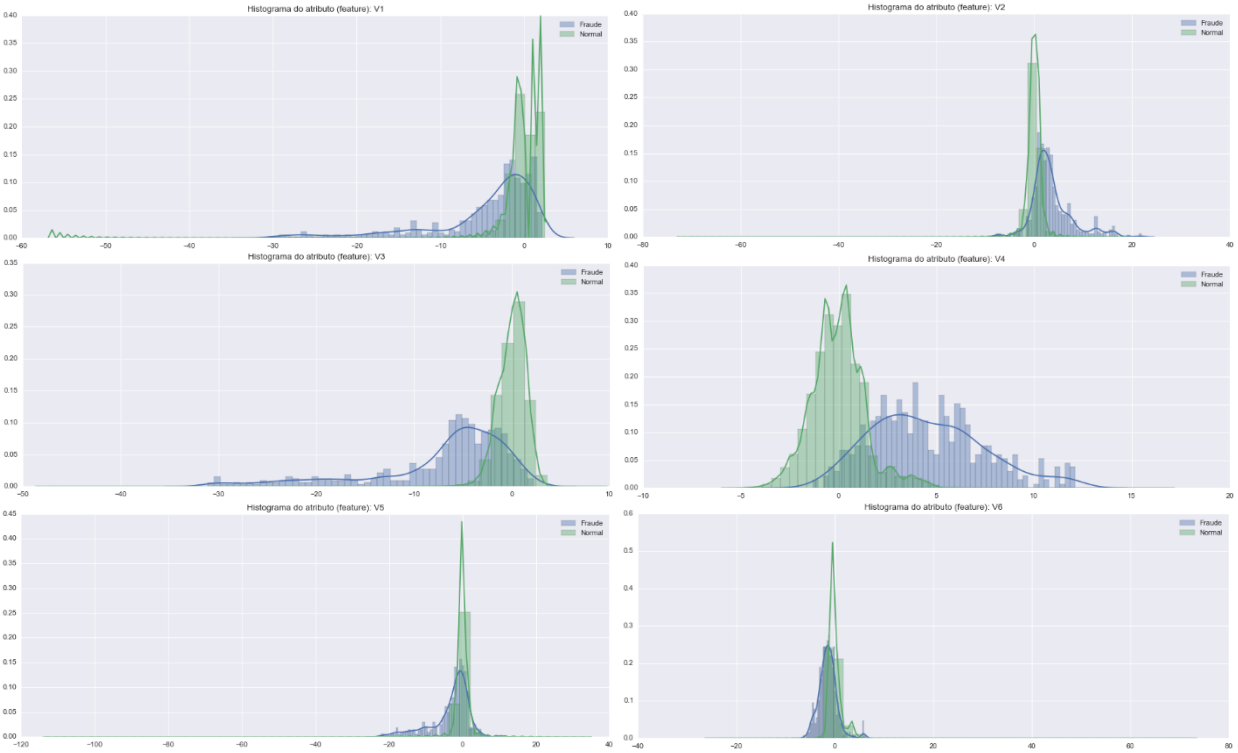


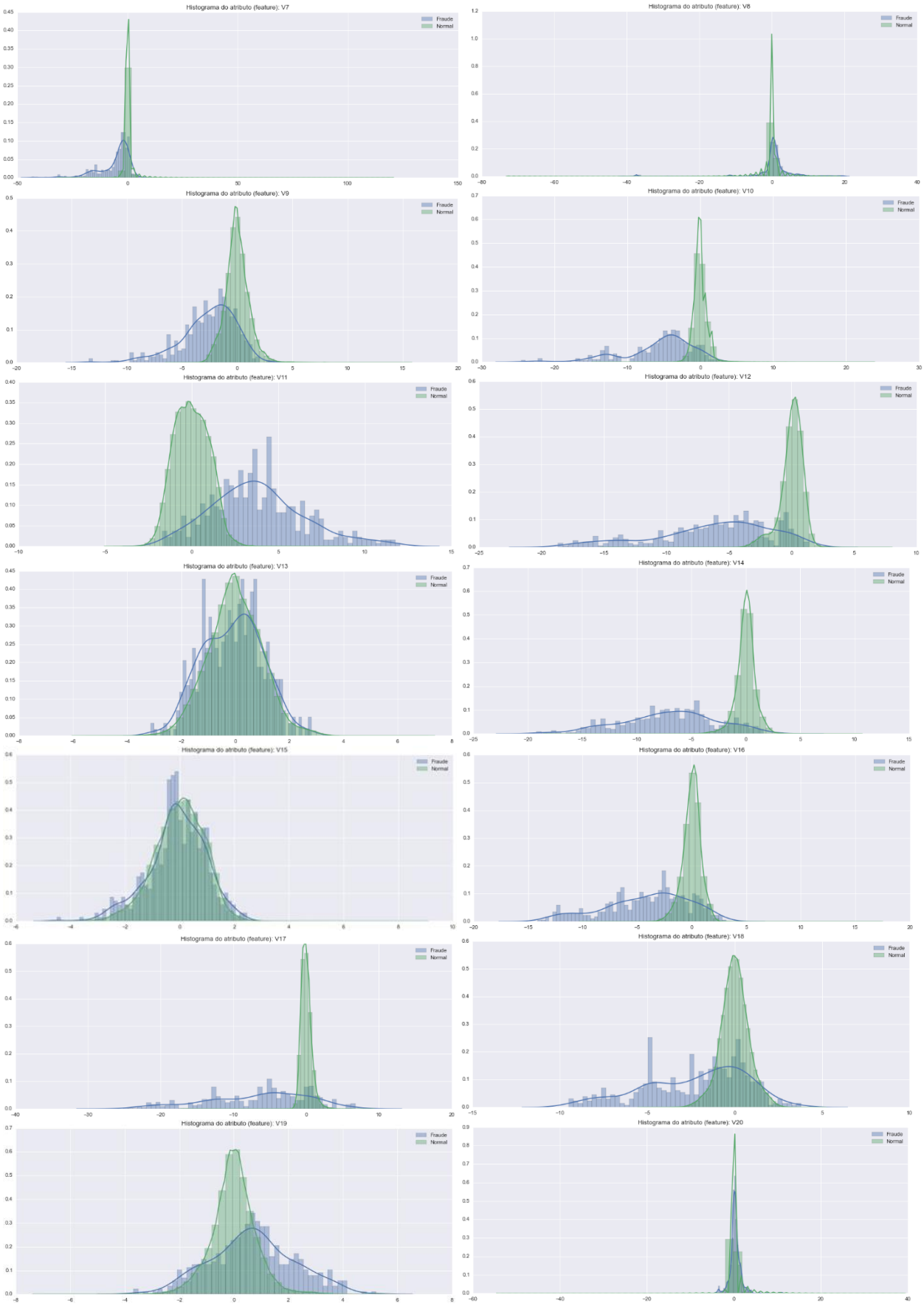
Gráfico 3 - Comparação do montante das transações ao longo do tempo

Nada de muito útil aqui.

B. Atributos mais significativos

De seguida vamos analisar as distribuições de cada um dos atributos anónimos (features) para tentar perceber se são todos significativos ou se existem alguns atributos que não são significativos.




Histograma do atributo (feature): V7
Histograma do atributo (feature): V8
Histograma do atributo (feature): V9
Histograma do atributo (feature): V10
Histograma do atributo (feature): V11
Histograma do atributo (feature): V12
Histograma do atributo (feature): V13
Histograma do atributo (feature): V14
Histograma do atributo (feature): V15
Histograma do atributo (feature): V16
Histograma do atributo (feature): V17
Histograma do atributo (feature): V18
Histograma do atributo (feature): V19
Histograma do atributo (feature): V20
Fraude
Normal

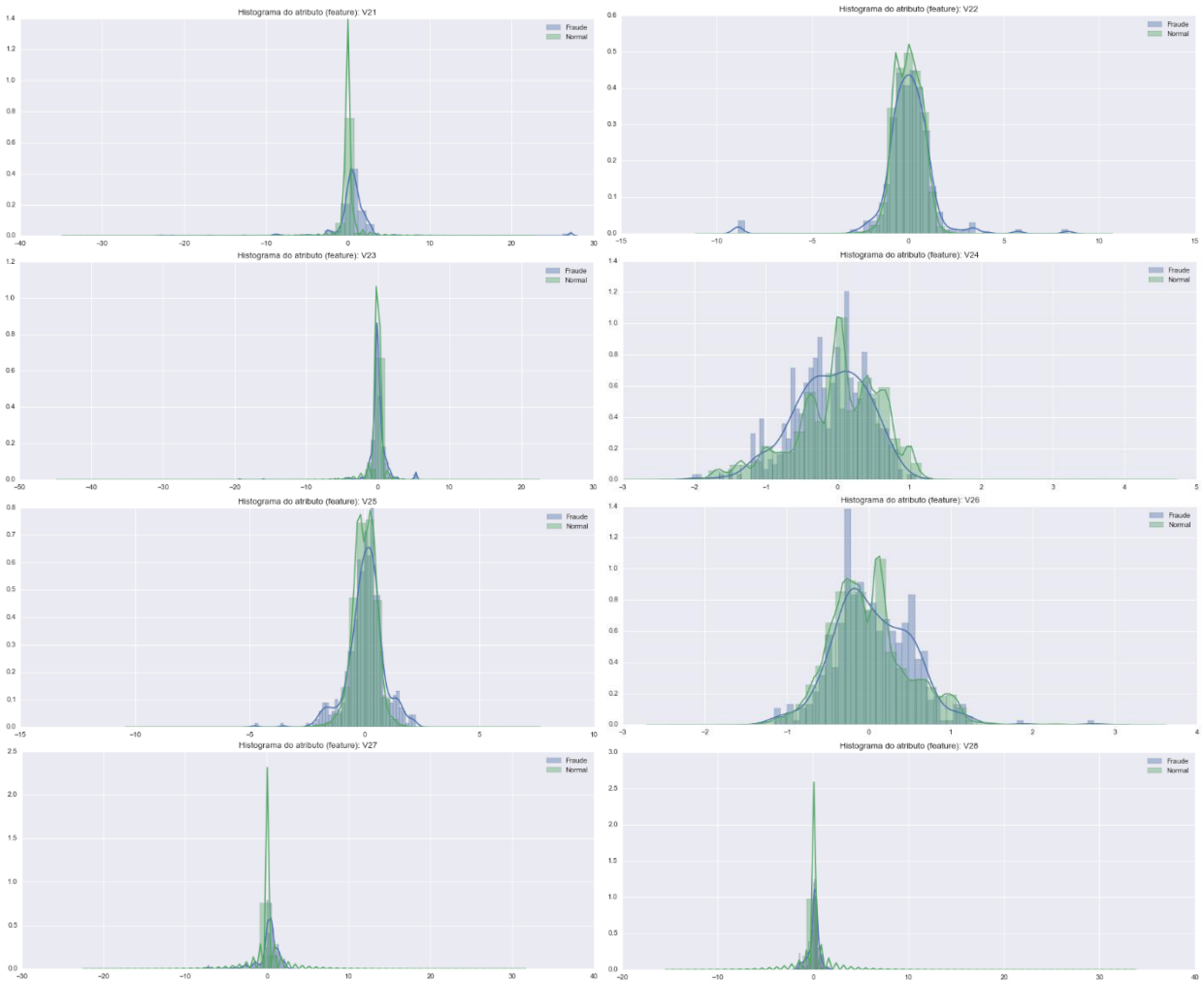


Gráfico 4 - Análise das distribuições de cada um dos atributos

Os atributos que têm distribuições muito semelhantes entre os dois tipos de transações não interessam (não são significativos), são eles: V8, V15, V20, V22, V23, V24, V25, V26, V27, V28.

Abaixo encontra-se o código fonte em Python utilizado na análise exploratória dos dados.

```python
import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.gridspec as gridspec
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE

def main():
    df = pd.read_csv('C:\\Users\\Rafael\\Downloads\\creditcard.csv')

    print(df.head())
    print(df.describe())
    print(df.isnull().sum())

    print('Fraude')
    print(df.Time[df.Class == 1].describe())
    print()
    print('Normal')
    print(df.Time[df.Class == 0].describe())

    f, (ax1, ax2) = plt.subplots(2, 1, sharex = True, figsize = (12, 4))
    ax1.hist([n for n in df.Time[df.Class == 1]], bins = 50)
    ax1.set_title('Fraude')
    ax2.hist([n for n in df.Time[df.Class == 0]], bins = 50)
    ax2.set_title('Normal')
    plt.xlabel('Tempo (em segundos)')
    plt.ylabel('Número de transações')
    plt.show()

    print('Fraude')
    print(df.Amount[df.Class == 1].describe())
    print()
    print('Normal')
    print(df.Amount[df.Class == 0].describe())

    f, (ax1, ax2) = plt.subplots(2, 1, sharex = True, figsize = (12, 4))
    ax1.hist([n for n in df.Amount[df.Class == 1]], bins = 30)
    ax1.set_title('Fraude')
    ax2.hist([n for n in df.Amount[df.Class == 0]], bins = 30)
    ax2.set_title('Normal')
    plt.xlabel('Montante (€)')
    plt.ylabel('Número de transações')
    plt.yscale('log')
    plt.show()

    df['Amount_max_fraud'] = 1
    df.loc[df.Amount <= 2125.87, 'Amount_max_fraud'] = 0

    f, (ax1, ax2) = plt.subplots(2, 1, sharex = True, figsize = (12, 6))
    ax1.scatter(df.Time[df.Class == 1], df.Amount[df.Class == 1])
    ax1.set_title('Fraude')
    ax2.scatter(df.Time[df.Class == 0], df.Amount[df.Class == 0])
    ax2.set_title('Normal')
    plt.xlabel('Tempo (em segundos)')
    plt.ylabel('Montante (€)')
    plt.show()

    for feature in df.ix[:,1:29].columns:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        sns.distplot(df[feature][df.Class == 1], bins = 50, label = 'Fraude')
        sns.distplot(df[feature][df.Class == 0], bins = 50, label = 'Normal')
        ax.set_xlabel('')
        ax.set_title('Histograma do atributo (feature): ' + str(feature))
        plt.legend(loc = 'best')
        plt.show()


if __name__ == '__main__':
    main()
```

Figura 1 - Código da análise exploratória dos dados

IV. Algoritmos da versão escalável do classificador

Nesta secção apresentamos os códigos dos classificadores desenvolvidos para este trabalho na linguagem de programação scala, de modo a permitir a sua execução paralela em vários nós. Devido às limitações da API do spark, apenas foi possível implementar três dos cinco classificadores apresentados no estado da arte, são eles as árvores de decisão, as florestas aleatórias e as árvores gradient-boosted. Abaixo segue-se o código para cada um dos classificadores.

1. Árvores de decisão

```
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.evaluation.MulticlassMetrics

// Carregar e fazer o parse do ficheiro de dados
val data = MLUtils.loadLibSVMFile(sc, "C:\\spark\\creditcard_MLUtils.csv")
// Dividir os dados em conjuntos de treino e de teste (30% para teste)
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

// Treinar um modelo da árvore de decisão
// categoricalFeaturesInfo vazio indica que todas as features são contínuas
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 5
val maxBins = 32

val start = System.currentTimeMillis
val model = DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,
  impurity, maxDepth, maxBins)
val tempo_treino = System.currentTimeMillis - start

start = System.currentTimeMillis
// Avaliar o modelo em instâncias de teste e calcula o erro de teste
val labelAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val tempo_classificacao = System.currentTimeMillis - start

val testErr = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
println("Test Error = " + testErr)
println("Learned classification tree model:\n" + model.toDebugString)
```

Figura 2 - Código do classificador baseado em árvores de decisão

O código acima começa por carregar os dados do ficheiro para o RDD nomeado por 'data'. Após o carregamento, os dados são divididos em duas partes, 70% dos dados para treino e os restantes 30% para teste. De seguida é construído um modelo a partir dos dados de treino para o classificador baseado em árvores de decisão. Por último o modelo é avaliado com os dados de teste, e é calculado o erro.

Para os dois algoritmos seguintes (florestas aleatórias e árvores Gradient-Boosted) a lógica é muito semelhante.

2. Florestas aleatórias

```
import org.apache.spark.mllib.tree.RandomForest
import org.apache.spark.mllib.tree.model.RandomForestModel
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.evaluation.MulticlassMetrics

// Carregar e fazer o parse do ficheiro de dados
val data = MLUtils.loadLibSVMFile(sc, "C:\\spark\\creditcard_MLUtils.csv")
// Dividir os dados em conjuntos de treino e de teste (30% para teste)
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

// Treinar um modelo das florestas aleatórias
// categoricalFeaturesInfo vazio indica que todas as features são contínuas
val numClasses = 2
val categoricalFeaturesInfo = Map[Int, Int]()
val numTrees = 3
val featureSubsetStrategy = "auto" // Deixa o algoritmo escolher
val impurity = "gini"
val maxDepth = 4
val maxBins = 32

val start = System.currentTimeMillis
val model = RandomForest.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,
  numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)
val tempo_treino = System.currentTimeMillis - start

start = System.currentTimeMillis
// Avaliar o modelo em instâncias de teste e calcula o erro de teste
val labelAndPreds = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val tempo_classificacao = System.currentTimeMillis - start

val testErr = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
println("Test Error = " + testErr)
println("Learned classification forest model:\n" + model.toDebugString)
```

Figura 3 - Código do classificador baseado em florestas aleatórias

3. Árvores Gradient-Boosted

```
import org.apache.spark.mllib.tree.GradientBoostedTrees
import org.apache.spark.mllib.tree.configuration.BoostingStrategy
import org.apache.spark.mllib.tree.model.GradientBoostedTreesModel
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.evaluation.MulticlassMetrics

// Carregar e fazer o parse do ficheiro de dados
val data = MLUtils.loadLibSVMFile(sc, "C:\\spark\\creditcard_MLUtils.csv")
// Dividir os dados em conjuntos de treino e de teste (30% para teste)
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, testData) = (splits(0), splits(1))

// Treinar um modelo de árvores com gradient-boosted
val boostingStrategy = BoostingStrategy.defaultParams("Classification")
boostingStrategy.numIterations = 3
boostingStrategy.treeStrategy.numClasses = 2
boostingStrategy.treeStrategy.maxDepth = 5
// categoricalFeaturesInfo vazio indica que todas as features são contínuas
boostingStrategy.treeStrategy.categoricalFeaturesInfo = Map[Int, Int]()

val start = System.currentTimeMillis
val model = GradientBoostedTrees.train(trainingData, boostingStrategy)
val tempo_treino = System.currentTimeMillis - start

start = System.currentTimeMillis
// Avaliar o modelo em instâncias de teste e calcular o erro de teste
val labelAndPreds = testData.map { point =>
    val prediction = model.predict(point.features)
    (point.label, prediction)
}
val tempo_classificacao = System.currentTimeMillis - start

val testErr = labelAndPreds.filter(r => r._1 != r._2).count().toDouble / testData.count()
println("Test Error = " + testErr)
println("Learned classification forest model:\n" + model.toDebugString)
```

Figura 4 - Código do classificador baseado em árvores gradient-boosted

4. Estatísticas dos classificadores

```
// --- Estatisticas (matriz de confusão, precisão)
val metrics = new MulticlassMetrics(labelAndPreds)

// Matriz de confusão
println("Matriz de confusao:")
println(metrics.confusionMatrix)

// Estatisticas gerais
println("Estatisticas sumarias")
println("Precisao = " + metrics.precision)
println("Recall = " + metrics.recall)
println("Tempo de treino = " + tempo_treino)
println("Tempo de classificacao = " + tempo_classificacao)
```

Figura 5 - Código para recolha de métricas dos classificadores

O código acima foi utilizado nos três classificadores para calcular a matriz de confusão, a precisão, o recall, o tempo de treino e o tempo de classificação.

V. Resultados experimentais e análises

Nesta secção iremos fazer uma comparação da precisão e do recall, assim como do desempenho de treino entre os diversos algoritmos de classificação de forma avaliar qual o melhor algoritmo de classificação para o problema de deteção de fraudes em cartões de crédito. Para além disso iremos fazer algumas experiências de escalabilidade com o objetivo de comparar o ganho de desempenho (speedup) entre os algoritmos de classificação pelo aumento do número de máquinas.

A. Comparação dos algoritmos de classificação

Nesta subsecção fizemos a comparação da precisão e do recall, assim como do desempenho de treino e classificação dos três tipos de classificadores (árvores de decisão, florestas aleatórias e árvores com gradient-boosted).

Para avaliarmos a precisão e o recall dos classificadores fizemos duas experiências, a primeira usando todas as features disponíveis no dataset e a segunda apenas com as features mais relevantes que foram estudadas durante a análise exploratória dos dados. Do dataset original utilizámos 70% dos registos para treinar os classificadores e 30% dos registos para teste. Com esses 30% de registos obtivemos a matriz de confusão, que nos permitiu calcular a precisão e o recall de fraude recorrendo às seguintes fórmulas:

- Precisão (fraude) = número de transações classificadas corretamente como fraude / número de transações classificadas como fraude
- Recall (fraude) = número de transações classificadas corretamente como fraude / número de fraudes reais

Para avaliarmos o desempenho de treino e de classificação dos classificadores replicámos o nosso ficheiro original de 169 Mb (284.807 transações) três vezes, dando origem a um novo ficheiro de 508 Mb (854.421 transações).

Para a realização destas experiências utilizámos uma única máquina com as seguintes caraterísticas:

- Sistema operativo: Ubuntu de 64 bits
- Processador: com 2 cores
- Memória RAM: 4 Gb

- Disco SSD com 100 GB

Experiência 1 - Comparação da precisão e do recall dos classificadores

Considerando todas as features do dataset (30 features), nas primeiras três tabelas abaixo apresentamos a matriz de confusão para cada um dos três tipos de classificadores. Na quarta tabela apresentamos um sumário com a precisão e o recall de cada classificador.

Matriz de confusão para o classificador baseado em árvores de decisão:

Realidade \ Classificação	Fraude	Não fraude
Fraude	101	23
Não fraude	37	85017

Matriz de confusão para o classificador baseado em florestas aleatórias:

Realidade \ Classificação	Fraude	Não fraude
Fraude	107	18
Não fraude	45	84790

Matriz de confusão para o classificador baseado em árvores gradient-boosted:

Realidade \ Classificação	Fraude	Não fraude
Fraude	113	12
Não fraude	42	85525

A tabela seguinte compara a precisão e o recall dos três algoritmos de classificação. Os valores foram calculados a partir da matriz de confusão de cada algoritmo recorrendo ás fórmulas da precisão e do recall anteriormente apresentadas.

Classificador	Precisão (%)	Recall (%)
Árvores de decisão	73.19	81.45
Florestas aleatórias	70.39	85.60
Árvores com gradient-boosted	72.90	90.40

Considerando apenas as features relevantes do dataset (um total de 20 features após a exclusão das features menos relevantes), nas primeiras três tabelas abaixo apresentamos a matriz de confusão para cada um dos três tipos de classificadores. Na quarta tabela apresentamos um sumário com a precisão e o recall de cada classificador.

Matriz de confusão para o classificador baseado em árvores de decisão:

Realidade \ Classificação	Fraude	Não fraude
Fraude	113	22
Não fraude	33	85906

Matriz de confusão para o classificador baseado em florestas aleatórias:

Realidade \ Classificação	Fraude	Não fraude
Fraude	106	19
Não fraude	33	85619

Matriz de confusão para o classificador baseado em árvores gradient-boosted:

Realidade \ Classificação	Fraude	Não fraude
Fraude	112	29
Não fraude	25	85250

A tabela seguinte compara a precisão e o recall dos três algoritmos de classificação. Os valores foram calculados a partir da matriz de confusão de cada algoritmo recorrendo ás fórmulas da precisão e do recall anteriormente apresentadas.

Classificador	Precisão (%)	Recall (%)
Árvores de decisão	77.40	83.70
Florestas aleatórias	76.26	84.80
Árvores com gradient-boosted	81.75	79.43

Os gráficos abaixo mostram a comparação da precisão e do recall dos vários classificadores nos dois modos, ou seja, considerando todas as features (30 features) e considerando apenas as features mais relevantes (20 features) do dataset.

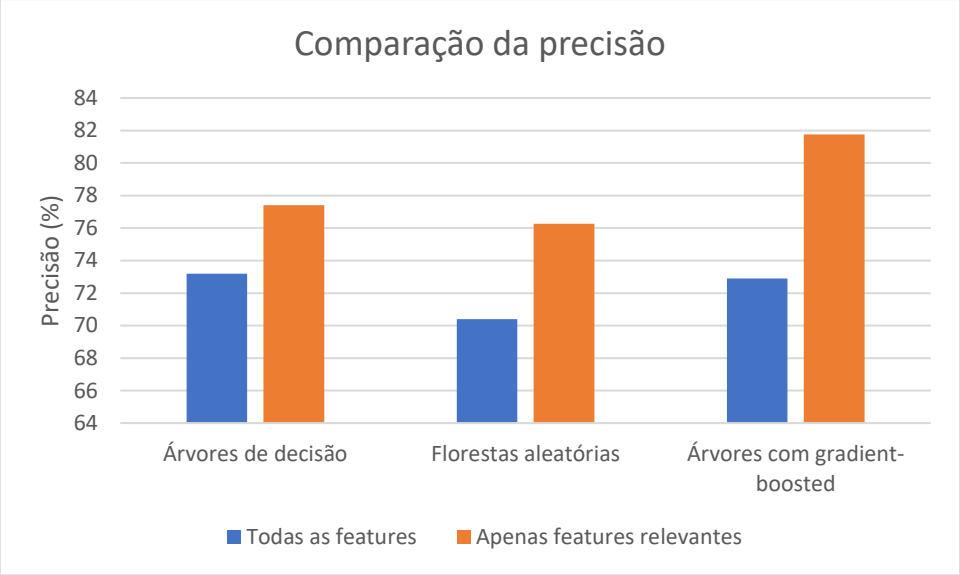


Gráfico 5 - Comparação da precisão dos classificadores

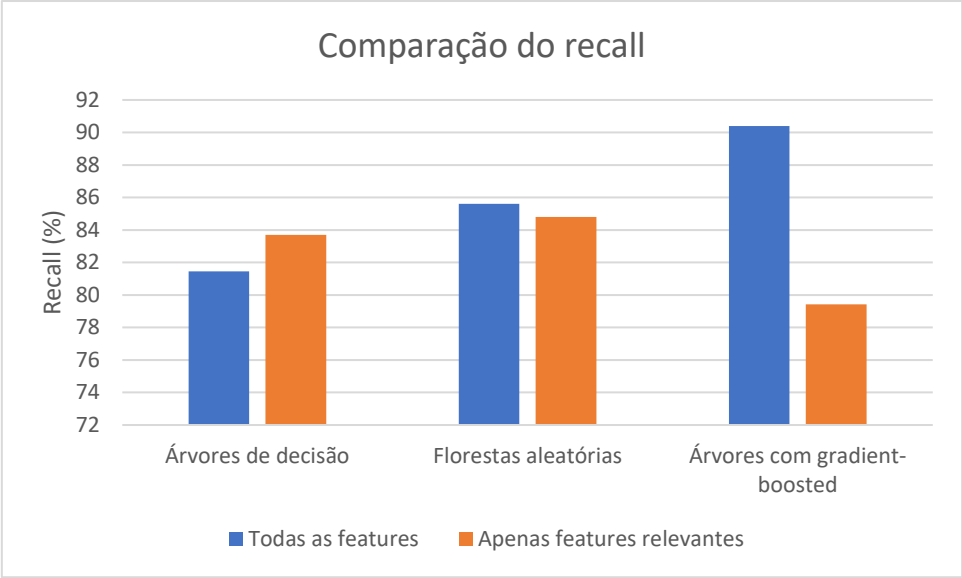


Gráfico 6 - Comparação do recall dos classificadores

A partir dos gráficos acima podemos concluir que o classificador baseado em árvores gradient-boosted obteve a melhor precisão, seguido das árvores de decisão e florestas aleatórias com uma precisão muito semelhante. De referir também um aumento significativo da precisão dos classificadores quando descartámos as features menos relevantes.

Quanto ao recall podemos verificar um aumento nos classificadores baseados em árvores de decisão e florestas aleatórias e uma ligeira diminuição no caso do classificador baseado em árvores gradient-boosted quando descartamos as features menos relevantes.

Tendo em consideração os valores da precisão e do recall obtidos, podemos concluir que as árvores com gradient-boosted foi o classificador que obteve o melhor resultado.

Experiência 2 - Comparação do desempenho dos classificadores

A tabela seguinte compara o desempenho dos algoritmos de classificação em termos de treino e classificação, quando executados em uma só máquina.

Classificador	Tempo de treino (s)	Tempo de classificação (s)
Árvores de decisão	7.532	0.609
Florestas aleatórias	6.969	0.391
Árvores com gradient-boosted	20.720	0.594

A partir da tabela acima podemos concluir que as florestas aleatórias foi o algoritmo com o melhor desempenho tanto em termos de treino como de classificação. Apesar do classificador baseado em árvores gradient-boosted ter obtido um excelente valor de precisão na análise anterior, nesta análise verifica-se que precisa de cerca de três vezes mais tempo para a construção do modelo de classificação quando comparado aos outros classificadores.

B. Testes de escalabilidade

Nesta subsecção iremos estudar o ganho de desempenho (speedup) dos classificadores conseguido pelo aumento do número de máquinas.

Para avaliarmos o speedup dos classificadores replicámos o nosso ficheiro original de 169 Mb (284.807 transações) trinta vezes, dando origem a um novo ficheiro com um tamanho de 5.086 Gb (8.544.210 transações).

Para esta experiência foram utilizadas máquinas com as seguintes caraterísticas:

- Sistema operativo: Ubuntu de 64 bits
- Processador: com 2 cores
- Memória RAM: 4 Gb
- Disco SSD com 100 GB

As duas tabelas seguintes mostram os tempos de treino e de classificação para cada classificador. A primeira tabela apenas em uma máquina, a segunda tabela em duas máquinas a executar em paralelo (em cluster).

Classificador - 1 máquina	Tempo de treino (s)	Tempo de classificação (s)
Árvores de decisão	599.290	1.587
Florestas aleatórias	623.118	1.704
Árvores com gradient-boosted	842.698	4.709

Classificador - 2 máquinas	Tempo de treino (s)	Tempo de classificação (s)
Árvores de decisão	288.246	1.252
Florestas aleatórias	272.373	1.245
Árvores com gradient-boosted	387.675	1.918

Nota: todos os tempos resultam da média de 3 medições efetuadas.

O gráfico abaixo compara o tempo de treino de cada classificador a executar em uma e em duas máquinas.

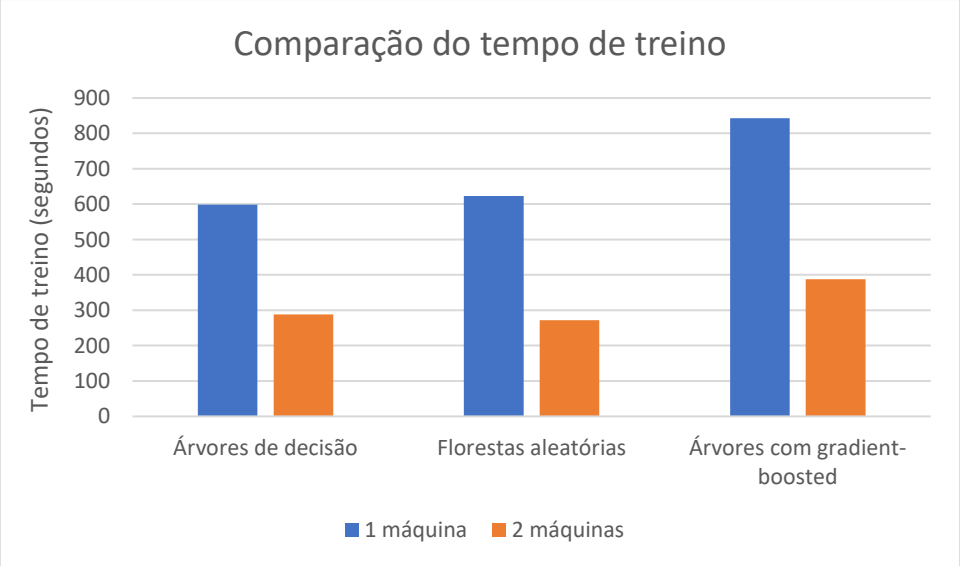


Gráfico 7 - Comparação do desempenho na construção dos classificadores

O gráfico abaixo compara o tempo de classificação das transações para cada classificador a executar em uma e em duas máquinas em simultâneo.

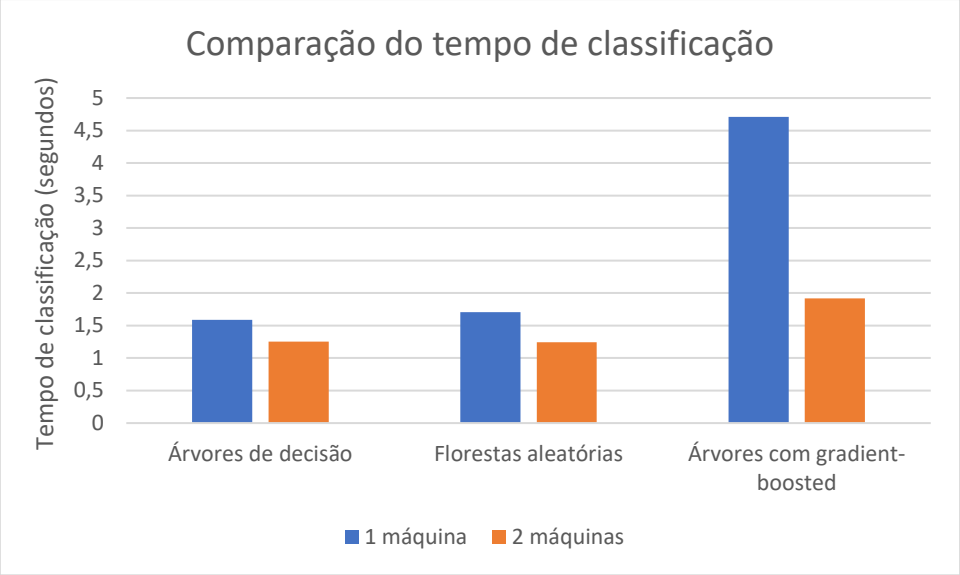


Gráfico 8 - Comparação do desempenho dos classificadores na classificação de transações

A tabela seguinte compara o speedup de treino e classificação dos algoritmos de classificação obtidos.

Classificador	Speedup treino	Speedup classificação
Árvores de decisão	2.08	1.27
Florestas aleatórias	2.29	1.37
Árvores com gradient-boosted	2.17	2.46

O gráfico abaixo compara de forma gráfica o speedup de treino e classificação de cada um dos classificadores obtido pela execução do algoritmo em cluster.

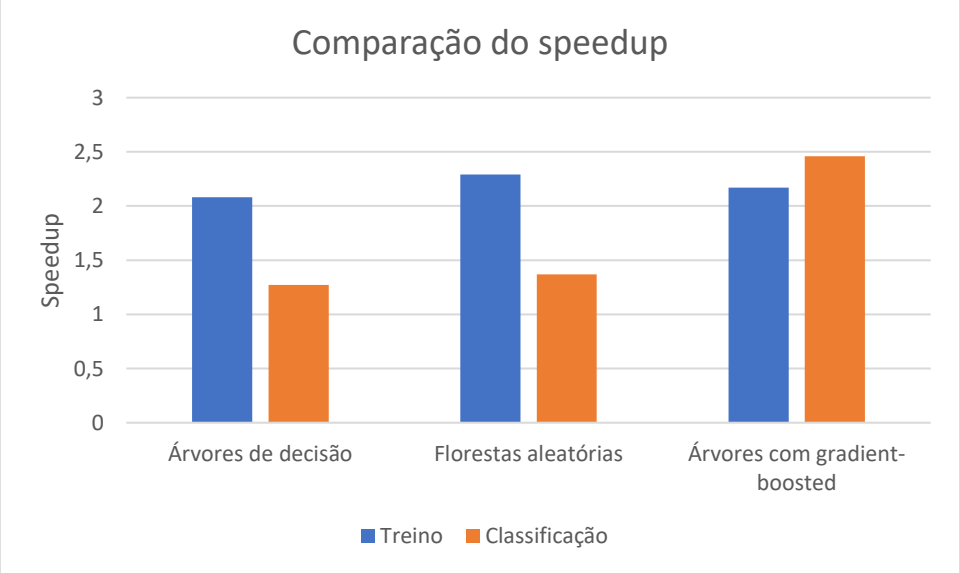


Gráfico 9 - Comparação do speedup de treino e classificação dos classificadores

Idealmente o speedup obtido pela adição de uma máquina deveria ser linear, no entanto isso não se verificou em todas as situações.
Em termos de treino do classificador conseguiu-se um speedup entre 2.08 e 2.29. Em termos de classificação um speedup máximo de 2.46 para o classificador baseado em árvores gradient-boosted.

VI. Conclusão

Este trabalho demostrou que é possível detetar fraudes em cartões de crédito com uma elevada precisão e com boa eficiência.
A partir do estudo das features mais relevantes das transações e do benchmarking dos algoritmos de classificação foi possível obter uma precisão máxima de 81.75% combinado com um recall de 84.80%. Em termos de precisão e recall o classificador que obteve melhores resultados foi as árvores com gradient-boosted. No entanto foi o classificador baseado em florestas aleatórias que obteve melhor desempenho tanto em termos de tempo de treino como tempo de classificação.
Com recurso ao apache spark estudámos o ganho de desempenho (speedup) dos classificadores conseguido pelo aumento do número de máquinas. Através das experiências efetuadas conseguiram-se speedups máximos de 2.29 no treino dos classificadores e de 2.46 na classificação de transações como legítimas ou fraudulentas.
Para trabalho futuro uma ideia interessante seria implementar um classificador baseado numa rede neuronal de convolução, que seja compatível com RDD's do spark, e comparar os resultados com os algoritmos já experimentados.

Referências

[1] https://www.kaggle.com/dalpozz/creditcardfraud
[2] https://spark.apache.org/
[3] https://spark.apache.org/docs/2.1.0/ml-classification-regression.html
[4] Maes, Sam, et al. "Credit card fraud detection using Bayesian and neural networks." Proceedings of the 1st international naiso congress on neuro fuzzy technologies. 2002.
Link: https://www.researchgate.net/profile/Karl_Tuyls/publication/254198382_Machine_Learning_Techniques_for_Fraud_Detection/links/555f695508ae6f4dcc926e88.pdf
[5] Raj, S. Benson Edwin, and A. Annie Portia. "Analysis on credit card fraud detection methods." Computer, Communication and Electrical Technology (ICCCET), 2011 International Conference on. IEEE, 2011.
Link: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5762457
[6] Bhattacharyya, Siddhartha, et al. "Data mining for credit card fraud: A comparative study." Decision Support Systems 50.3 (2011): 602-613.
Link: http://www.sciencedirect.com/science/article/pii/S0167923610001326
[7] Bahnsen, Alejandro Correa, et al. "Feature engineering strategies for credit card fraud detection." Expert Systems With Applications 51 (2016): 134-142.
Link: http://www.sciencedirect.com/science/article/pii/S0957417415008386