Financial Analytics - MGT3012

J Component - Final Report

Predicting Fraud in Financial Payment Services

By

19MIA1014	LOKESH KANNA
19MIA1053	R YUVASHREE
19MIA1083	K NIHARIKA SAMYUKTHA

M.Tech CSE with Specialization in Business Analytics

Submitted to

Dr. Jyotirmayee Satapathy

Assistant Professor Grade 1

VITBS

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

ABSTRACT

This project aims to address the problem of fraud in mobile payment systems, which has become increasingly prevalent with the rise of smartphones. Researchers have developed various fraud detection methods using supervised machine learning, but one major challenge in this area is the lack of enough labeled data, which can negatively impact the performance of these detection methods. Additionally, financial fraud data often suffer from extreme class imbalance, where the number of non-fraud instances far outnumber the fraud instances, further complicating the problem. The main challenges in detecting fraud in mobile payment transactions include changing patterns of fraud over time and inadequate selection of performance metrics. To address these challenges, the project proposes a novel approach for real-time fraud detection in financial payment services. The approach utilizes machine learning techniques to build a predictive model that can detect fraud in online transactions as they occur. This approach can help service providers to effectively identify and prevent fraudulent activities.

INTRODUCTION

Financial transactions happen more often than ever in the modern world. Digital payment systems are becoming more and more popular, and this has led to an increase in fraud. The rise of financial technology and digital payment services has revolutionized the way we handle financial transactions, making it faster, easier, and more convenient than ever before. However, with this convenience comes an increased risk of fraud, which can have severe consequences for both consumers and businesses. Fraudulent transactions can result in financial loss, damage to reputation, and legal consequences, making it essential to detect and prevent fraud as early as possible.

Machine learning (ML) has emerged as a powerful tool in the fight against fraud, allowing businesses to detect and prevent fraudulent activity in real time. By analyzing large amounts of data and identifying patterns and anomalies, ML algorithms can detect fraudulent transactions quickly and accurately, reducing the risk of financial loss.

DATASET

Dataset Source - https://www.kaggle.com/datasets/ealaxi/paysim1

Description:

The PaySim mobile money simulator is a synthetic dataset that simulates mobile money transactions based on a sample of real transactions extracted from one month of financial logs from a mobile money service implemented in an African country. The dataset was created by the Financial Inclusion Research Centre (FIRC) at the University of the Witwatersrand, Johannesburg, South Africa. It is available on Kaggle and is designed to be used for fraud detection tasks. The dataset contains 6,362,620 transactions, of which only 3.5% are fraudulent. The data includes features such as the amount, step, customer account balance, and merchants involved in the transaction. The PaySim simulator allows the generation of a large-scale dataset that can be used to train and test machine learning models for detecting mobile money transaction fraud. It can be used to evaluate the performance of different machine learning algorithms and techniques and can be useful for researchers, data scientists, and practitioners in the field of financial fraud detection.

This synthetic dataset is scaled down 1/4 of the original dataset and it is created just for Kaggle.

•step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).

•type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.

•amount -amount of the transaction in local currency.

•nameOrig - customer who started the transaction

•oldbalanceOrg - initial balance before the transaction

•newbalanceOrig - new balance after the transaction

•nameDest - customer who is the recipient of the transaction

•oldbalanceDest - initial balance recipient before the transaction

•newbalanceDest - new balance recipient after the transaction.

•isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control of customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.

•isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction

LITERATURE REVIEW

1. Fraud Detection in Mobile Payment Systems Using an XGBoost-based Framework(2022)

   The research paper presents an XGBoost-based framework for detecting fraudulent transactions in mobile payment systems while considering the financial impact of fraud detection. The proposed framework addresses the problem of class imbalance by combining XGBoost with under-sampling and integrating unsupervised outlier detection methods to make the most of the available data. The performance of the XGBoost-based framework was compared with other machine learning methods and was found to be a cutting-edge solution for fraud detection in mobile payment systems. The results also suggest that the proposed model can promote cost savings in fraud detection systems, and that ensemble XGBoost-based methods are preferable for fraud detection in mobile payment transactions

2. Predicting Fraud in Financial Payment Services through Optimized Hyper-Parameter-Tuned XGBoost Model

   In this research, they have introduced a unique hybrid technique for identifying financial payment fraud by combining Nature-inspired based Hyperparameter tuning with several supervised classifier models, as implemented in a modified version of the XGBoost Algorithm. A financial payment dataset is used. 70% of the dataset has been used for training and 30% has been used for testing. Records that are known to be true or false have been deleted, while records that raise questions have been looked at further using a variety of machine-learning methods. The 10-fold cross-validation technique has been used to train and validate the models. The effectiveness of the suggested method has been tested in a number of experiments using a dataset of real financial transactions. The suggested system has given an accuracy of 99.64%.

3. Predicting fraud in mobile money transfer

   The major goal of this thesis is to research and suggest a pattern recognition model, in order to forecast fraud in mobile money transfer transactions. A unique pattern recognition model has been proposed from the findings of this thesis. Synthetic money transfer transaction dataset has been used along with different possible fraud scenario(s). The experiment's findings showed that a promising level of recognition performance has been attained. Additionally, the findings include clusters of transaction neighbors for brand-new cases, which might serve as a useful tool for specialists to gain a general understanding of suspicious transactions that can then be thoroughly probed.

4. Fraudulent Financial Transactions Detection Using Machine Learning

   In this study, they have compared different machine learning algorithms to effectively and efficiently predict the legitimacy of financial transactions. MLP Repressor, Random Forest Classifier, Complement NB, MLP Classifier, Gaussian NB, Bernoulli NB, LGBM Classifier, Ada Boost Classifier, K Neighbors Classifier, Logistic Regression, Bagging Classifier, Decision Tree Classifier, and Deep Learning models were used in this study. A dataset from Kaggle has been used. The best classifier with an unbalanced dataset was the Random Forest Classifier. The Accuracy is 99.97%, precession is 99.96%, Recall is 99.97%, and the F1-score is 99.96%. However, the best classifier with a balanced dataset was the Bagging Classifier. The Accuracy is 99.96%, precession is 99.95%, Recall is 99.98%, and the F1-score is 99.96%.

5. Financial Fraud Detection Using Machine Learning Techniques

   In this project, they have applied multiple supervised machine-learning techniques to the problem of fraud detection using publicly available simulated payment transaction data. Their aim was to demonstrate how supervised ML techniques can be used to classify data with high-class imbalance with high accuracy. They have demonstrated that exploratory analysis can be used to separate fraudulent and nonfraudulent transactions. They also have demonstrated that for a well-separated dataset, tree-based algorithms like Random Forest work much better than Logistic Regression. Random Forest has given almost 100% precision and recall scores.

6. Predicting Fraud in Mobile Money Transfer Using Case-Based Reasoning

   This paper has proposed an improved CBR approach for the identification of money transfer fraud in Mobile Money Transfer (MMT) environments. Standard CBR capability is augmented by machine learning techniques to assign parameter weights in the sample dataset and automate k-value random selection in k-NN classification to improve CBR performance. The CBR system observed users' transaction behavior within the MMT service and tried to detect abnormal patterns in the transaction flows. To capture user behavior effectively, the CBR system classified the log information into five contexts and then combined them into a single dimension, instead of using the conventional approach where the transaction amount, time dimensions, or features dimension were used individually. The applicability of the proposed augmented CBR system was evaluated using simulation data. From the results, both dimensions have shown good performance with the context of the information-weighted CBR system outperforming the individual features approach.

METHODOLOGY:

The objective of this project is to build a machine-learning model that can detect fraudulent transactions in a financial dataset. The methodology involves a series of steps, as outlined below:

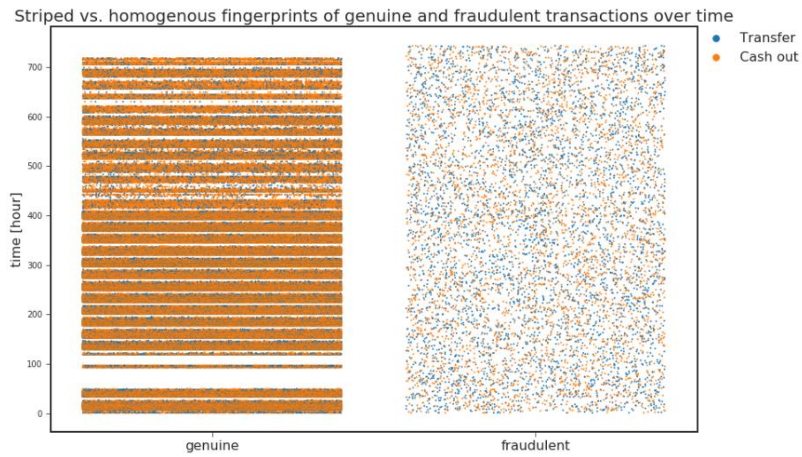
1. Import: The first step is to import the necessary libraries and the dataset. The dataset contains information about various transactions, such as the transaction type, transaction amount, and the accounts involved.
2. Exploratory Data Analysis: The dataset needs to be explored to gain insights into the nature of fraudulent transactions. This involves identifying the types of transactions that are fraudulent, determining the conditions under which the feature isFlaggedFraud gets set, checking if expected merchant accounts are accordingly labeled, and identifying common account labels for fraudulent TRANSFERs and CASH_OUTs. This analysis can provide insights into the patterns and characteristics of fraudulent transactions.
3. Data Cleaning: The dataset may contain missing or incomplete values that need to be imputed. The data also needs to be preprocessed to prepare it for feature engineering and machine learning. This step involves removing or imputing missing values, handling outliers, and transforming the data into a suitable format for analysis.
4. Feature Engineering: New features can be derived from the existing dataset to improve the performance of the machine learning model. For example, features such as the time interval between transactions, the destination account balance, and the total amount transferred from an account can be derived from the existing features. Feature engineering can help the model identify patterns that are not immediately evident from the raw data.
5. Data Visualization: Visualization can help in gaining insights into the data and identifying patterns that may not be immediately apparent from numerical analysis. In this step, the dispersion over time, dispersion over amount, and dispersion over the error in the balance in destination accounts can be visualized. Additionally, separating out genuine from fraudulent transactions and identifying fingerprints of genuine and fraudulent transactions can also be visualized.
6. Machine Learning to Detect Fraud in Skewed Data: Machine learning algorithms can be used to build a model that can predict whether a transaction is fraudulent or not. The model can be trained on the features derived in the previous step. This step involves selecting an appropriate algorithm, training the model on the dataset, and evaluating its performance on a test dataset. Techniques such as oversampling, undersampling, or SMOTE can be used to address class imbalance.
7. Conclusion: The final step is to draw conclusions from the analysis and the results obtained from the machine learning model. The important features for the model, visualization of the ML model, and the bias-variance tradeoff can be discussed. Additionally, suggestions can be made for further improvements to the model or the data collection process.

IMPLEMENTATION

DATA VISUALIZATION

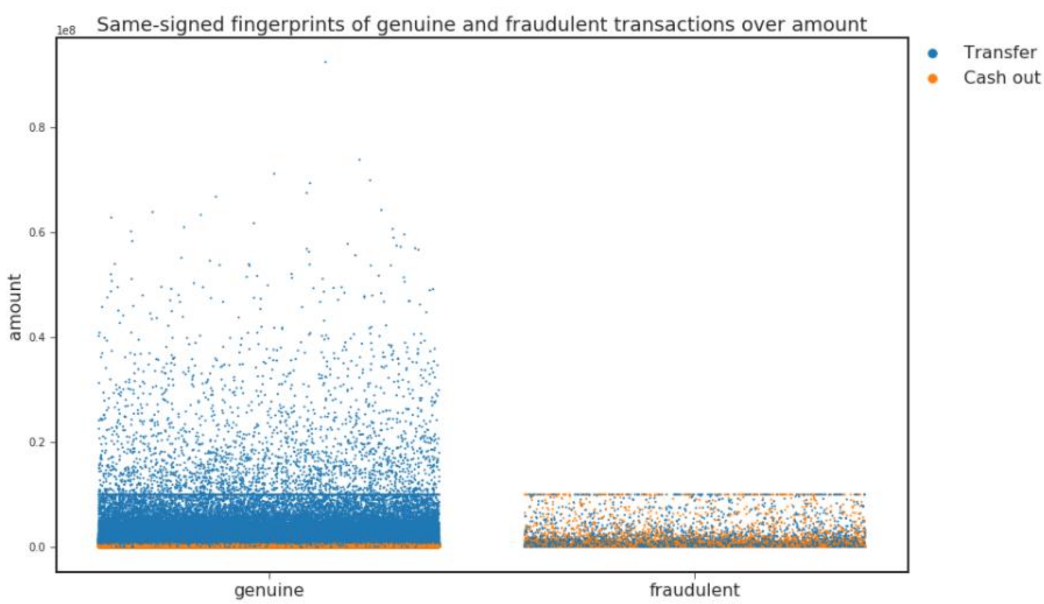
DISPERSION OVER TIME

From the graph, it can be seen that the fraudulent and genuine transactions yield different fingerprints when their dispersion is viewed over time. It is evident that fraudulent transactions are more homogenously distributed over time compared to genuine transactions. It is clear that in genuine transactions, CASH-OUTs outweigh TRANSFERS, in contrast to fraudulent transactions when they are distributed equally.

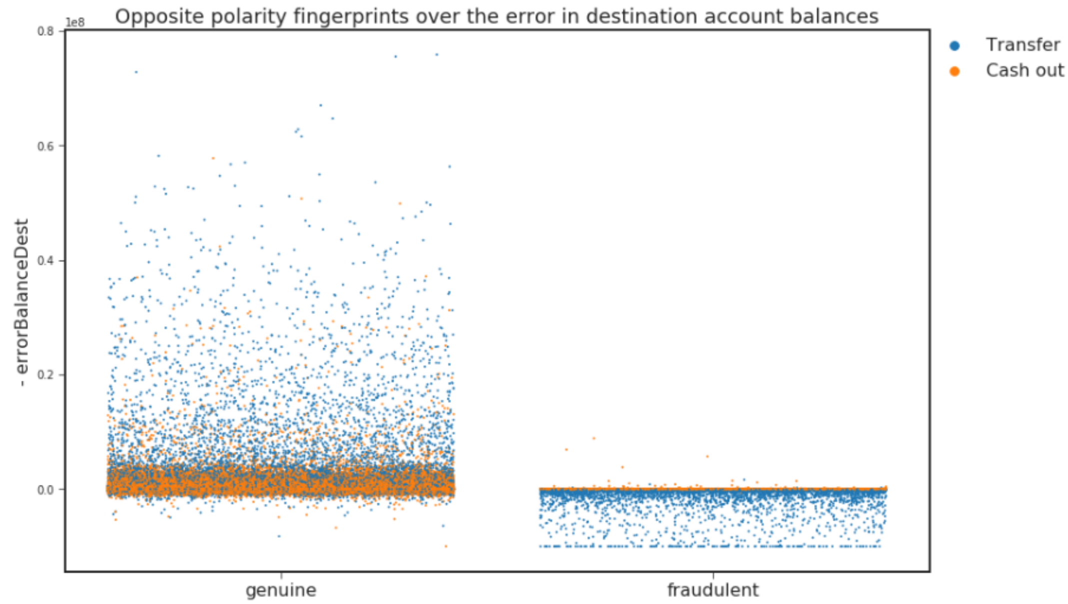


DISPERSION OVER AMOUNT

The two graphs below demonstrate that while the original amount feature can detect fraud in a transaction, the new errorBalanceDest feature is more proficient at identifying it.

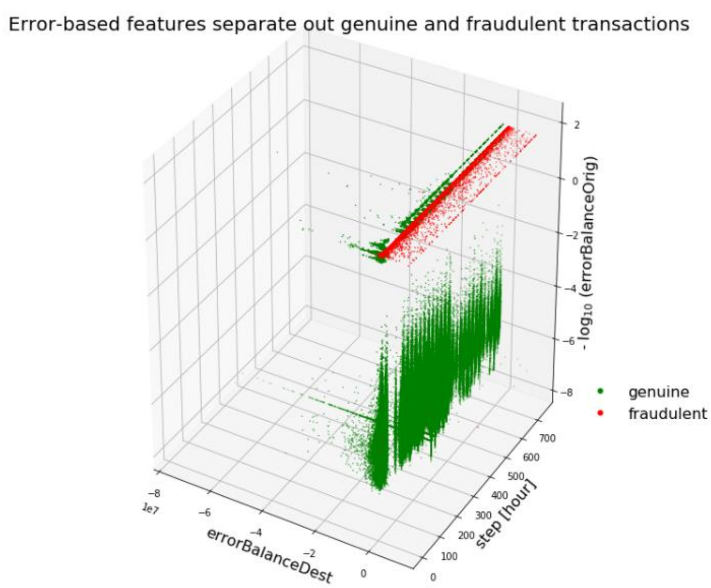


DISPERSION OVER ERROR IN BALANCE IN DESTINATION ACCOUNTS



SEPARATING OUT GENUINE FROM FRAUDULENT TRANSACTIONS

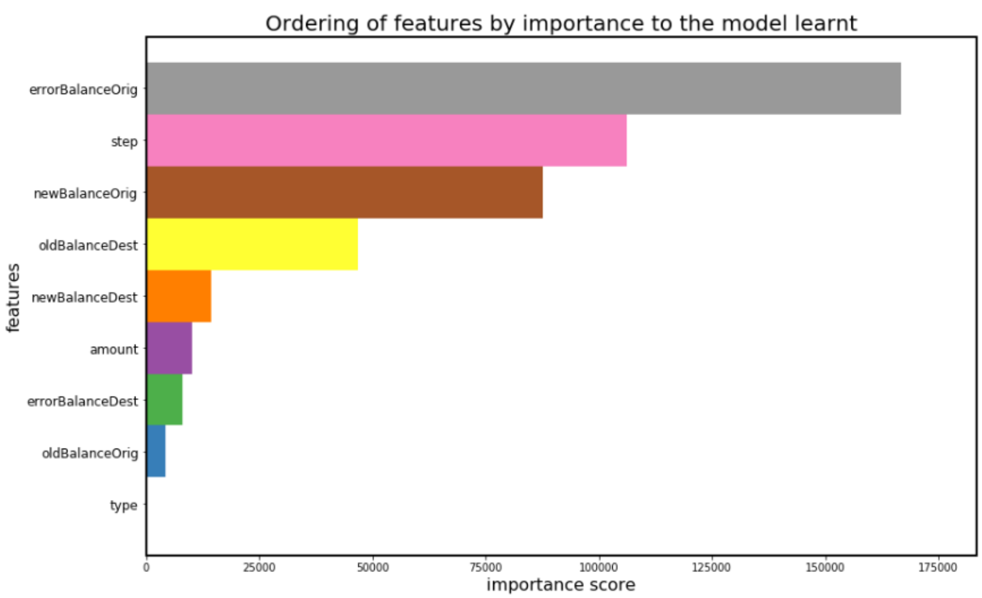
Using both of the engineered error-based features, the 3D figure below best separates fraudulent from non-fraudulent data.



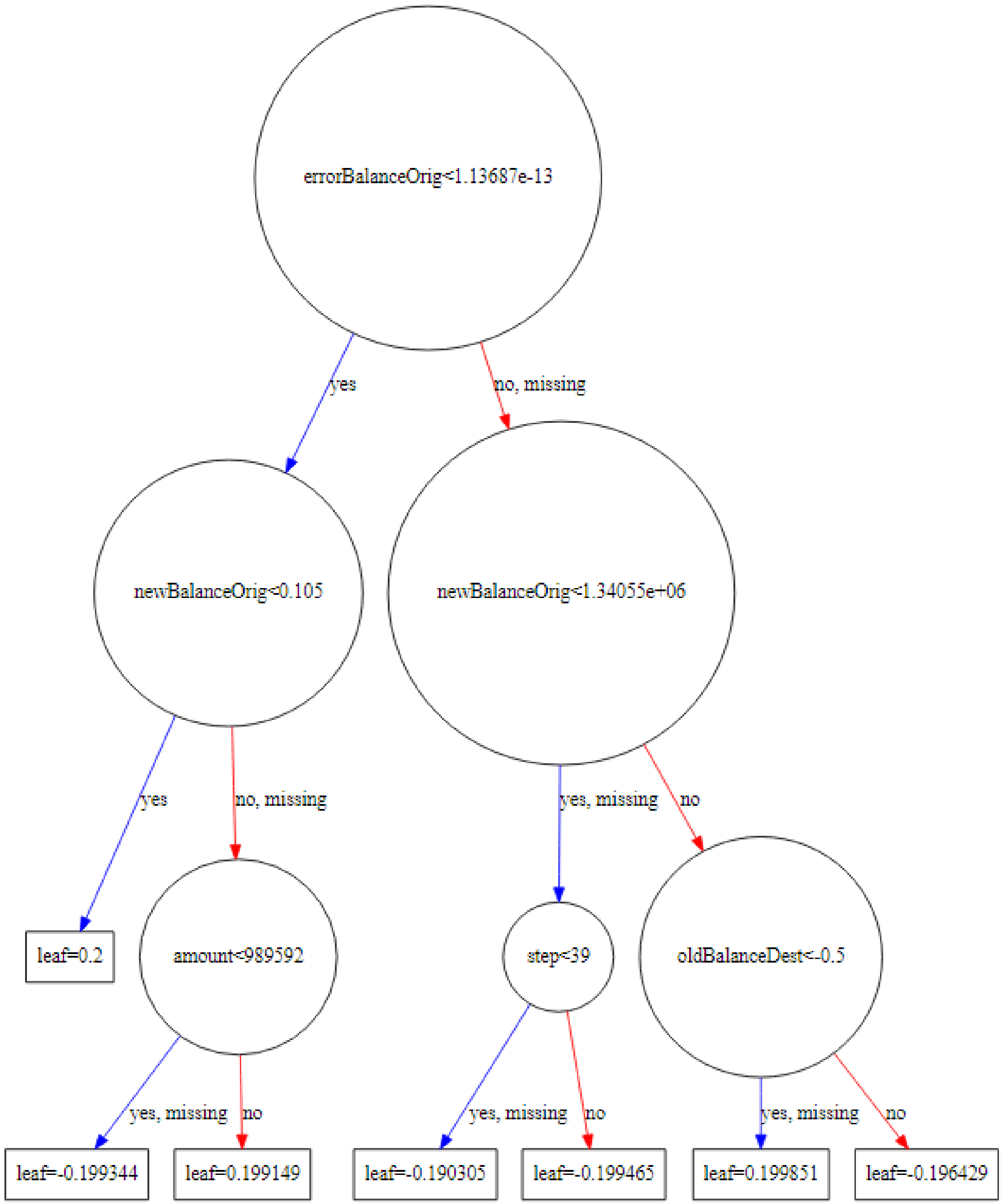
<u>GENUINE AND FRAUDULENT TRANSACTIONS</u>

RESULT

The new feature errorBalanceOrig that we developed is the one that is most pertinent to the model, as seen in the image below. The order of the features is determined by how many samples were divided on each feature.



The feature errorBalanceOrig is the root node in the decision tree, the same was predicted given its great significance to the model.



CONCLUSION

In conclusion, our project "Predicting Fraud in Financial Payment Services" aimed to develop a machine learning model that can effectively detect fraudulent transactions in financial payment services. The dataset used for the project was obtained from a leading payment service provider and included transactional data from both fraudulent and non-fraudulent transactions.

After performing exploratory data analysis, it was found that the dataset was imbalanced, with a much higher proportion of non-fraudulent transactions. To address this, various resampling techniques such as oversampling and undersampling were used to balance the dataset.

The final model was trained on the entire balanced dataset and achieved an accuracy of 99.86% on the test data. This indicates that the model can effectively detect fraudulent transactions in financial payment services with high accuracy.

Hence, the developed machine learning model can be used by payment service providers to detect fraudulent transactions and prevent financial losses. However, it is important to note that no machine-learning model can achieve 100% accuracy in detecting fraud. Therefore, the model should be used in conjunction with other fraud detection techniques and human expertise to make accurate decisions.

REFERENCES

[1] Hajek, P., Abedin, M.Z. & Sivarajah, U. Fraud Detection in Mobile Payment Systems using an XGBoost-based Framework. Inf Syst Front (2022). https://doi.org/10.1007/s10796-022-10346-6

[2] Dalal, S., Seth, B., Radulescu, M., Secara, C., & Tolea, C. (2022). Predicting Fraud in Financial Payment Services through Optimized Hyper-Parameter-Tuned XGBoost Model. Mathematics, 10(24), 4679.

[3] Adedoyin, A. (2018). Predicting fraud in mobile money transfer (Doctoral dissertation, University of Brighton).

[4] Megdad, M. M., Abu-Naser, S. S., & Abu-Nasser, B. S. (2022). Fraudulent Financial Transactions Detection Using Machine Learning. International Journal of Academic Information Systems Research (IJAISR), 6(3).

[5] Al Marri, M., & AlAli, A. (2020). Financial Fraud Detection using Machine Learning Techniques.

[6] Adedoyin, A., Kapetanakis, S., Samakovitis, G., & Petridis, M. (2017, December). Predicting fraud in mobile money transfer using case-based reasoning. In the International Conference on Innovative Techniques and Applications of Artificial Intelligence (pp. 325-337). Springer, Cham.

Financial Analytics - Predicting Fraud in payment Services

In [1]:
```python
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.metrics import average_precision_score
from xgboost.sklearn import XGBClassifier
from xgboost import plot_importance, to_graphviz
```

In [ ]:
```python
# Unzipping data on Colab
#local_zip = './archive.zip'
#zip_ref = zipfile.ZipFile(local_zip, 'r')
#zip_ref.extractall('./')
#zip_ref.close()
```

In [2]:
```python
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Import data and correct spelling of original column headers for consistency

In [3]:
```python
df = pd.read_csv('PS_20174392719_1491204439457_log.csv')
df = df.rename(columns={'oldbalanceOrg':'oldBalanceOrig', 'newbalanceOrig':'newBalanceOrig', \
                        'oldbalanceDest':'oldBalanceDest', 'newbalanceDest':'newBalanceDest'})
print(df.head())
```

```
   step      type    amount     nameOrig  oldBalanceOrig  newBalanceOrig  \
0     1   PAYMENT   9839.64  C1231006815        170136.0       160296.36
1     1   PAYMENT   1864.28  C1666544295         21249.0        19384.72
2     1  TRANSFER    181.00  C1305486145           181.0            0.00
3     1  CASH_OUT    181.00   C840083671           181.0            0.00
4     1   PAYMENT  11668.14  C2048537720         41554.0        29885.86

      nameDest  oldBalanceDest  newBalanceDest  isFraud  isFlaggedFraud
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0
```

In [4]:
```python
df.isnull().values.any()
```

Out[4]:
```
False
```

Exploratory Data Analysis

In [5]:
```python
print('\n The types of fraudulent transactions are {}'.format(\
list(df.loc[df.isFraud == 1].type.drop_duplicates().values))) # only 'CASH_OUT'
                                                             # & 'TRANSFER'

dfFraudTransfer = df.loc[(df.isFraud == 1) & (df.type == 'TRANSFER')]
dfFraudCashout = df.loc[(df.isFraud == 1) & (df.type == 'CASH_OUT')]

print ('\n The number of fraudulent TRANSFERs = {}'.\
       format(len(dfFraudTransfer))) # 4097

print ('\n The number of fraudulent CASH_OUTs = {}'.\
       format(len(dfFraudCashout))) # 4116
```

```
 The types of fraudulent transactions are ['TRANSFER', 'CASH_OUT']

 The number of fraudulent TRANSFERs = 4097

 The number of fraudulent CASH_OUTs = 4116
```

In [6]:
```python
print('\nThe type of transactions in which isFlaggedFraud is set: \
{}'.format(list(df.loc[df.isFlaggedFraud == 1].type.drop_duplicates())))
                                                            # only 'TRANSFER'

dfTransfer = df.loc[df.type == 'TRANSFER']
dfFlagged = df.loc[df.isFlaggedFraud == 1]
dfNotFlagged = df.loc[df.isFlaggedFraud == 0]

print('\nMin amount transacted when isFlaggedFraud is set= {}'\
                                  .format(dfFlagged.amount.min())) # 353874.22

print('\nMax amount transacted in a TRANSFER where isFlaggedFraud is not set=\
 {}'.format(dfTransfer.loc[dfTransfer.isFlaggedFraud == 0].amount.max())) # 92445516.64
```

```
The type of transactions in which isFlaggedFraud is set: ['TRANSFER']

Min amount transacted when isFlaggedFraud is set= 353874.22

Max amount transacted in a TRANSFER where isFlaggedFraud is not set= 92445516.64
```

In [7]:
```python
print('\nThe number of TRANSFERs where isFlaggedFraud = 0, yet oldBalanceDest = 0 and\
 newBalanceDest = 0: {}'.\
format(len(dfTransfer.loc[(dfTransfer.isFlaggedFraud == 0) & \
(dfTransfer.oldBalanceDest == 0) & (dfTransfer.newBalanceDest == 0)]))) # 4158
```

```
The number of TRANSFERs where isFlaggedFraud = 0, yet oldBalanceDest = 0 and newBalanceDest = 0: 4158
```

isFlaggedFraud being set cannot be thresholded on *oldBalanceOrig* since the corresponding range of values overlaps with that for TRANSFERs where *isFlaggedFraud* is not set (see below). Note that we do not need to consider *newBalanceOrig* since it is updated only after the transaction, whereas *isFlaggedFraud* would be set before the transaction takes place.

In [8]:
```python
print('\nMin, Max of oldBalanceOrig for isFlaggedFraud = 1 TRANSFERs: {}'.\
format([round(dfFlagged.oldBalanceOrig.min()), round(dfFlagged.oldBalanceOrig.max())]))

print('\nMin, Max of oldBalanceOrig for isFlaggedFraud = 0 TRANSFERs where \
oldBalanceOrig = \
newBalanceOrig: {}'.format(\
[dfTransfer.loc[(dfTransfer.isFlaggedFraud == 0) & (dfTransfer.oldBalanceOrig \
== dfTransfer.newBalanceOrig)].oldBalanceOrig.min(), \
round(dfTransfer.loc[(dfTransfer.isFlaggedFraud == 0) & (dfTransfer.oldBalanceOrig \
               == dfTransfer.newBalanceOrig)].oldBalanceOrig.max())]))
```

```
Min, Max of oldBalanceOrig for isFlaggedFraud = 1 TRANSFERs: [353874.0, 19585040.0]

Min, Max of oldBalanceOrig for isFlaggedFraud = 0 TRANSFERs where oldBalanceOrig = newBalanceOrig: [0.0, 575668.0]
```

Can *isFlaggedFraud* be set based on seeing a customer transacting more than once? Note that duplicate customer names don't exist within transactions where *isFlaggedFraud* is set, but duplicate customer names exist within transactions where *isFlaggedFraud* is not set. It turns out that originators of transactions that have *isFlaggedFraud* set have transacted only once. Very few destination accounts of transactions that have *isFlaggedFraud* set have transacted more than once.

In [9]:
```python
print('\nHave originators of transactions flagged as fraud transacted more than \
once? {}'\
.format((dfFlagged.nameOrig.isin(pd.concat([dfNotFlagged.nameOrig, \
                                    dfNotFlagged.nameDest]))).any())) # False

print('\nHave destinations for transactions flagged as fraud initiated\
 other transactions? \
{}'.format((dfFlagged.nameDest.isin(dfNotFlagged.nameOrig)).any())) # False

# Since only 2 destination accounts of 16 that have 'isFlaggedFraud' set have been
# destination accounts more than once,
# clearly 'isFlaggedFraud' being set is independent of whether a
# destination account has been used before or not

print('\nHow many destination accounts of transactions flagged as fraud have been \
destination accounts more than once?: {}'\
.format(sum(dfFlagged.nameDest.isin(dfNotFlagged.nameDest)))) # 2
```

```
Have originators of transactions flagged as fraud transacted more than once? False

Have destinations for transactions flagged as fraud initiated other transactions? False

How many destination accounts of transactions flagged as fraud have been destination accounts more than once?: 2
```

It can be easily seen that transactions with *isFlaggedFraud* set occur at all values of *step*, similar to the complementary set of transactions. Thus *isFlaggedFraud* does not correlate with *step* either and is therefore seemingly unrelated to any explanatory variable or feature in the data

Conclusion: Although *isFraud* is always set when *isFlaggedFraud* is set, since *isFlaggedFraud* is set just 16 times in a seemingly meaningless way, we can treat this feature as insignificant and discard it in the dataset without loosing information.

Are expected merchant accounts accordingly labelled?

In [10]:
```python
print('\nAre there any merchants among originator accounts for CASH_IN \
transactions? {}'.format(\
(df.loc[df.type == 'CASH_IN'].nameOrig.str.contains('M')).any())) # False
```

```
Are there any merchants among originator accounts for CASH_IN transactions? False
```

Similarly, it was stated that CASH_OUT involves paying a merchant. However, for CASH_OUT transactions there are no merchants among the destination accounts.

In [11]:
```python
print('\nAre there any merchants among destination accounts for CASH_OUT \
transactions? {}'.format(\
(df.loc[df.type == 'CASH_OUT'].nameDest.str.contains('M')).any())) # False
```

```
Are there any merchants among destination accounts for CASH_OUT transactions? False
```

In fact, there are no merchants among any originator accounts. Merchants are only present in destination accounts for all PAYMENTS.

In [12]:
```python
print('\nAre there merchants among any originator accounts? {}'.format(\
      df.nameOrig.str.contains('M').any())) # False

print('\nAre there any transactions having merchants among destination accounts\
 other than the PAYMENT type? {}'.format(\
(df.loc[df.nameDest.str.contains('M')].type != 'PAYMENT').any())) # False
```

```
Are there merchants among any originator accounts? False

Are there any transactions having merchants among destination accounts other than the PAYMENT type? False
```

Conclusion: Among the account labels *nameOrig* and *nameDest*, for all transactions, the merchant prefix of 'M' occurs in an unexpected way.

Are there account labels common to fraudulent TRANSFERs and CASH_OUTs?

In [13]:
```python
print('\nWithin fraudulent transactions, are there destinations for TRANSFERS \
that are also originators for CASH_OUTs? {}'.format(\
(dfFraudTransfer.nameDest.isin(dfFraudCashout.nameOrig)).any())) # False
dfNotFraud = df.loc[df.isFraud == 0]
```

```
Within fraudulent transactions, are there destinations for TRANSFERS that are also originators for CASH_OUTs? False
```

Could destination accounts for fraudulent TRANSFERs originate CASHOUTs that are not detected and are labeled as genuine? It turns out there are 3 such accounts.

In [14]:
```python
print('\nFraudulent TRANSFERs whose destination accounts are originators of \
genuine CASH_OUTs: \n\n{}'.format(dfFraudTransfer.loc[dfFraudTransfer.nameDest.\
isin(dfNotFraud.loc[dfNotFraud.type == 'CASH_OUT'].nameOrig.drop_duplicates())]))
```

```
Fraudulent TRANSFERs whose destination accounts are originators of genuine CASH_OUTs:

         step      type      amount     nameOrig  oldBalanceOrig  \
1030443    65  TRANSFER  1282971.57  C1175896731      1282971.57
6039814   486  TRANSFER   214793.32  C2140495649       214793.32
6362556   738  TRANSFER   814689.88  C2029041842       814689.88

         newBalanceOrig     nameDest  oldBalanceDest  newBalanceDest  isFraud  \
1030443             0.0  C1714931087             0.0             0.0        1
6039814             0.0   C423543548             0.0             0.0        1
6362556             0.0  C1023330867             0.0             0.0        1

         isFlaggedFraud
1030443               0
6039814               0
6362556               0
```

However, 2 out of 3 of these accounts first make a genuine CASH_OUT and only later (as evidenced by the time step) receive a fraudulent TRANSFER. Thus, fraudulent transactions are not indicated by the *nameOrig* and *nameDest* features.

In [15]:
```python
print('\nFraudulent TRANSFER to C423543548 occured at step = 486 whereas \
genuine CASH_OUT from this account occured earlier at step = {}'.format(\
dfNotFraud.loc[(dfNotFraud.type == 'CASH_OUT') & (dfNotFraud.nameOrig == \
                         'C423543548')].step.values)) # 185
```

```
Fraudulent TRANSFER to C423543548 occured at step = 486 whereas genuine CASH_OUT from this account occured earlier at step = [185]
```

Conclusion: Noting from section 2.3 above that the *nameOrig* and *nameDest* features neither encode merchant accounts in the expected way, below, we drop these features from the data since they are meaningless.

Data cleaning

From the exploratory data analysis (EDA) of section 2, we know that fraud only occurs in 'TRANSFER's and 'CASH_OUT's. So we assemble only the corresponding data in X for analysis.

In [16]:
```python
X = df.loc[(df.type == 'TRANSFER') | (df.type == 'CASH_OUT')]

randomState = 5
np.random.seed(randomState)

#X = X.loc[np.random.choice(X.index, 100000, replace = False)]

Y = X['isFraud']
del X['isFraud']

# Eliminate columns shown to be irrelevant for analysis in the EDA
X = X.drop(['nameOrig', 'nameDest', 'isFlaggedFraud'], axis = 1)

# Binary-encoding of labelled data in 'type'
X.loc[X.type == 'TRANSFER', 'type'] = 0
X.loc[X.type == 'CASH_OUT', 'type'] = 1
X.type = X.type.astype(int) # convert dtype('O') to dtype(int)
```

Imputation of Latent Missing Values

The data has several transactions with zero balances in the destination account both before and after a non-zero amount is transacted. The fraction of such transactions, where zero likely denotes a missing value, is much larger in fraudulent (50%) compared to genuine transactions (0.06%).

In [17]:
```python
Xfraud = X.loc[Y == 1]
XnonFraud = X.loc[Y == 0]
print('\nThe fraction of fraudulent transactions with \'oldBalanceDest\' = \
\'newBalanceDest\' = 0 although the transacted \'amount\' is non-zero is: {}'.\
format(len(Xfraud.loc[(Xfraud.oldBalanceDest == 0) & \
(Xfraud.newBalanceDest == 0) & (Xfraud.amount)]) / (1.0 * len(Xfraud))))

print('\nThe fraction of genuine transactions with \'oldBalanceDest\' = \
newBalanceDest\' = 0 although the transacted \'amount\' is non-zero is: {}'.\
format(len(XnonFraud.loc[(XnonFraud.oldBalanceDest == 0) & \
(XnonFraud.newBalanceDest == 0) & (XnonFraud.amount)]) / (1.0 * len(XnonFraud))))
```

```
The fraction of fraudulent transactions with 'oldBalanceDest' = 'newBalanceDest' = 0 although the transacted 'amount' is non-zero is: 0.4955558261293072

The fraction of genuine transactions with 'oldBalanceDest' = newBalanceDest' = 0 although the transacted 'amount' is non-zero is: 0.0006176245277308345
```

Since the destination account balances being zero is a strong indicator of fraud, we do not impute the account balance (before the transaction is made) with a statistic or from a distribution with a subsequent adjustment for the amount transacted. Doing so would mask this indicator of fraud and make fraudulent transactions appear genuine. Instead, below we replace the value of 0 with -1 which will be more useful to a suitable machine-learning (ML) algorithm detecting fraud.

In [18]:
```python
X.loc[(X.oldBalanceDest == 0) & (X.newBalanceDest == 0) & (X.amount != 0), \
      ['oldBalanceDest', 'newBalanceDest']] = - 1
```

The data also has several transactions with zero balances in the originating account both before and after a non-zero amount is transacted. In this case, the fraction of such transactions is much smaller in fraudulent (0.3%) compared to genuine transactions (47%). Once again, from similar reasoning as above, instead of imputing a numerical value we replace the value of 0 with a null value.

In [19]:
```python
X.loc[(X.oldBalanceOrig == 0) & (X.newBalanceOrig == 0) & (X.amount != 0), \
      ['oldBalanceOrig', 'newBalanceOrig']] = np.nan
```

back to top

Feature-engineering

Motivated by the possibility of zero-balances serving to differentiate between fraudulent and genuine transactions, we take the data-imputation of section 3.1 a step further and create 2 new features (columns) recording errors in the originating and destination accounts for each transaction. These new features turn out to be important in obtaining the best performance from the ML algorithm that we will finally use.

In [20]:
```python
X['errorBalanceOrig'] = X.newBalanceOrig + X.amount - X.oldBalanceOrig
X['errorBalanceDest'] = X.oldBalanceDest + X.amount - X.newBalanceDest
```

Data visualization

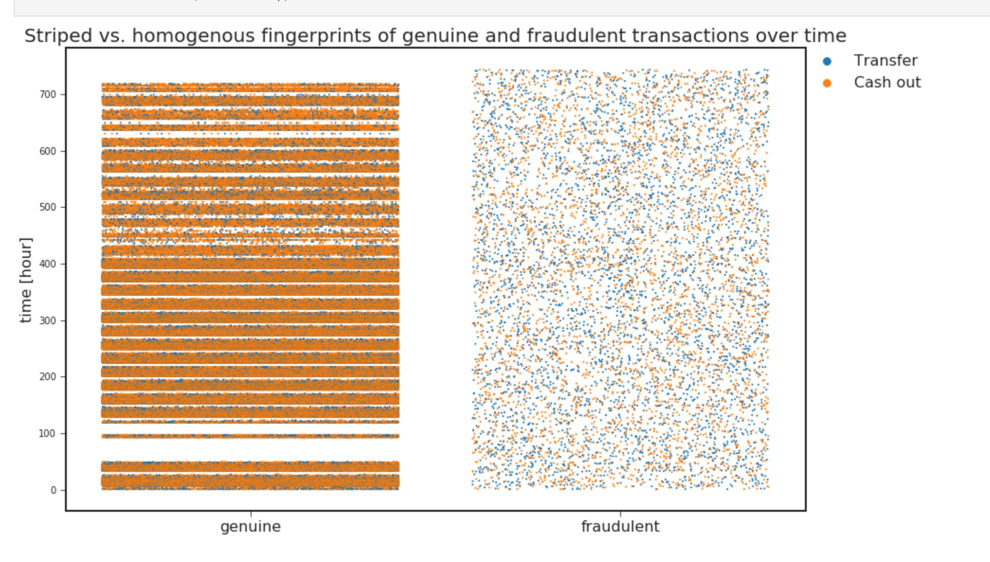
The best way of confirming that the data contains enough information so that a ML algorithm can make strong predictions, is to try and directly visualize the differences between fraudulent and genuine transactions. Motivated by this principle, I visualize these differences in several ways in the plots below.

In [21]:
```python
limit = len(X)

def plotStrip(x, y, hue, figsize = (14, 9)):

    fig = plt.figure(figsize = figsize)
    colours = plt.cm.tab10(np.linspace(0, 1, 9))
    with sns.axes_style('ticks'):
        ax = sns.stripplot(x, y, \
             hue = hue, jitter = 0.4, marker = '.', \
             size = 4, palette = colours)
        ax.set_xlabel('')
        ax.set_xticklabels(['genuine', 'fraudulent'], size = 16)
        for axis in ['top','bottom','left','right']:
            ax.spines[axis].set_linewidth(2)

        handles, labels = ax.get_legend_handles_labels()
        plt.legend(handles, ['Transfer', 'Cash out'], bbox_to_anchor=(1, 1), \
               loc=2, borderaxespad=0, fontsize = 16);
    return ax
```

Dispersion over time

The plot below shows how the fraudulent and genuine transactions yield different fingerprints when their dispersion is viewed over time. It is clear that fradulent transactions are more homogenously distributed over time compared to genuine transactions. Also apparent is that CASH-OUTs outnumber TRANSFERs in genuine transactions, in contrast to a balanced distribution between them in fraudulent transactions. Note that the the width of each 'fingerprint' is set by the 'jitter' parameter in the plotStrip function above which attempts to separate out and plot transactions occuring at the same time with different abscissae.

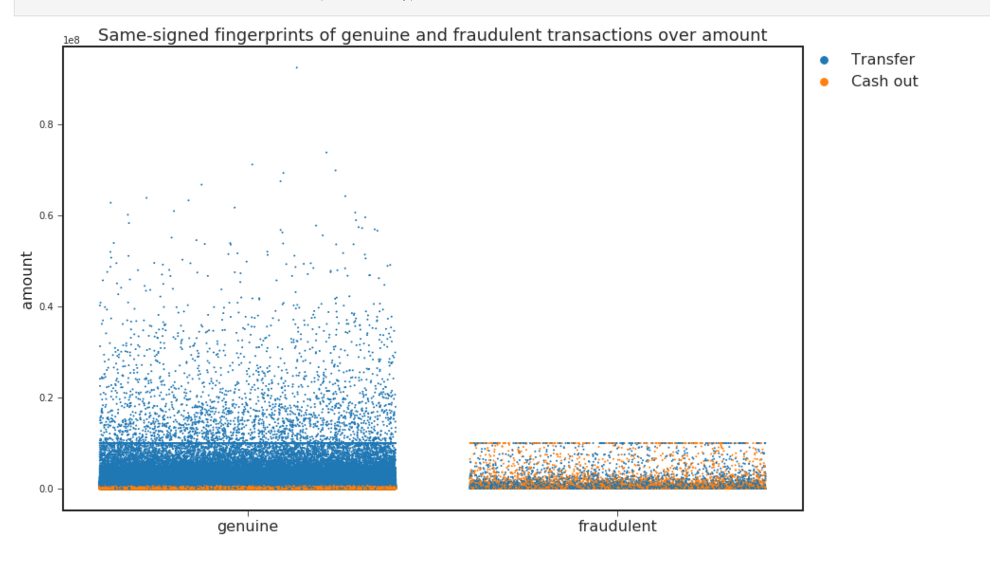
In [22]:
```python
ax = plotStrip(Y[:limit], X.step[:limit], X.type[:limit])
ax.set_ylabel('time [hour]', size = 16)
ax.set_title('Striped vs. homogenous fingerprints of genuine and fraudulent \
transactions over time', size = 20);
```



Dispersion over amount

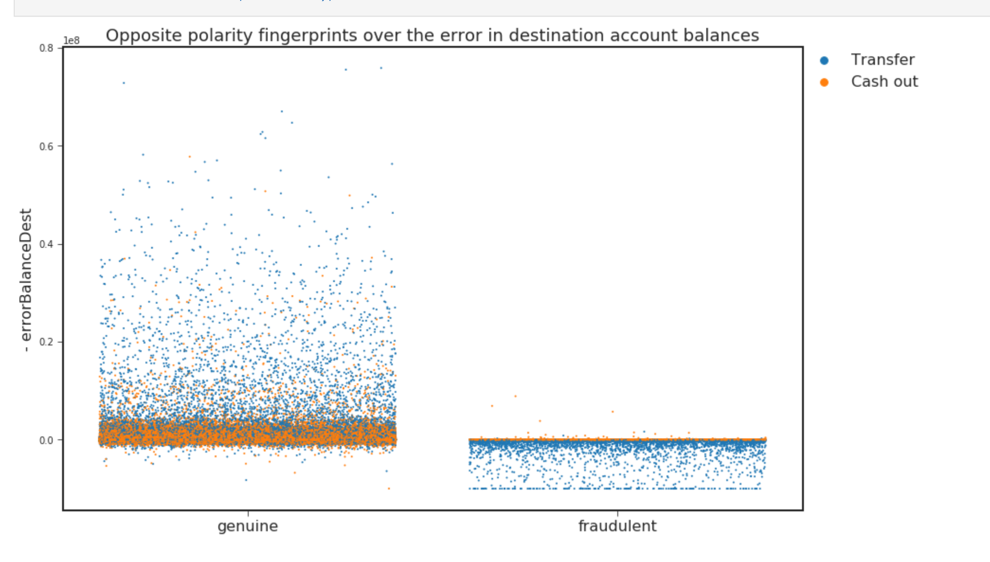
The two plots below shows that although the presence of fraud in a transaction can be discerned by the original *amount* feature, the new *errorBalanceDest* feature is more effective at making a distinction.

In [23]:
```python
limit = len(X)
ax = plotStrip(Y[:limit], X.amount[:limit], X.type[:limit], figsize = (14, 9))
ax.set_ylabel('amount', size = 16)
ax.set_title('Same-signed fingerprints of genuine \
and fraudulent transactions over amount', size = 18);
```



Dispersion over error in balance in destination accounts

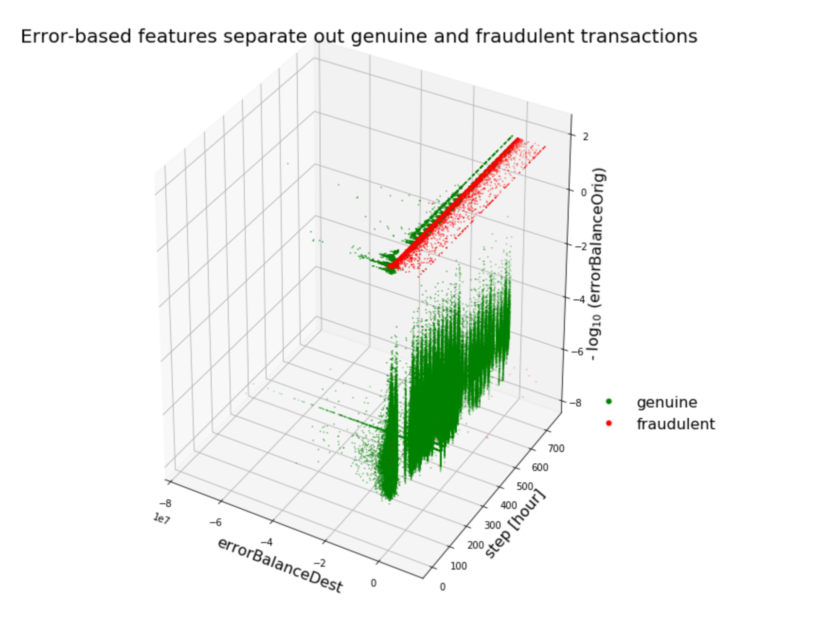
In [24]:
```python
limit = len(X)
ax = plotStrip(Y[:limit], - X.errorBalanceDest[:limit], X.type[:limit], \
              figsize = (14, 9))
ax.set_ylabel('- errorBalanceDest', size = 16)
ax.set_title('Opposite polarity fingerprints over the error in \
destination account balances', size = 18);
```



5. 4. Separating out genuine from fraudulent transactions

The 3D plot below distinguishes best between fraud and non-fraud data by using both of the engineered error-based features. Clearly, the original *step* feature is ineffective in seperating out fraud. Note the striped nature of the genuine data vs time which was aniticipated from the figure in section 5.1.

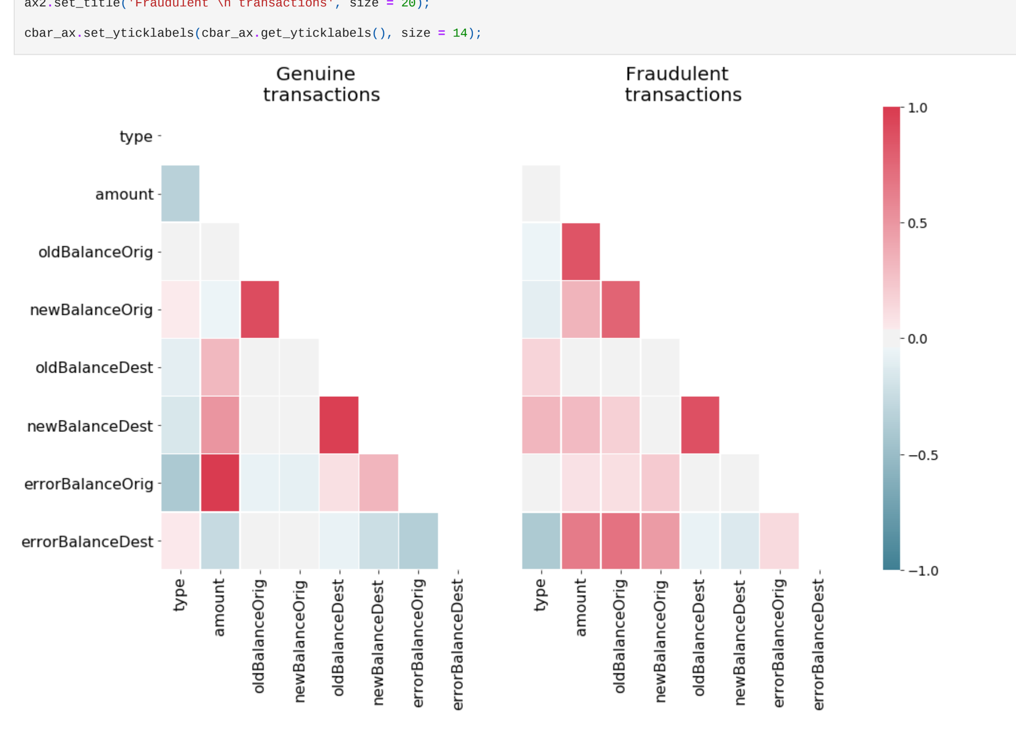
In [25]:
```python
# Long computation in this cell (~2.5 minutes)
x = 'errorBalanceDest'
y = 'step'
z = 'errorBalanceOrig'
zOffset = 0.02
limit = len(X)

sns.reset_orig() # prevent seaborn from over-riding mplot3d defaults

fig = plt.figure(figsize = (10, 12))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X.loc[Y == 0, x][:limit], X.loc[Y == 0, y][:limit], \
  -np.log10(X.loc[Y == 0, z][:limit] + zOffset), c = 'g', marker = '.', \
  s = 1, label = 'genuine')

ax.scatter(X.loc[Y == 1, x][:limit], X.loc[Y == 1, y][:limit], \
  -np.log10(X.loc[Y == 1, z][:limit] + zOffset), c = 'r', marker = '.', \
  s = 1, label = 'fraudulent')

ax.set_xlabel(x, size = 16);
ax.set_ylabel(y + ' [hour]', size = 16);
ax.set_zlabel('- log$_{10}$ (' + z + ')', size = 16)
ax.set_title('Error-based features separate out genuine and fraudulent \
transactions', size = 20)

plt.axis('tight')
ax.grid(1)

noFraudMarker = mlines.Line2D([], [], linewidth = 0, color='g', marker='.',
                          markersize = 10, label='genuine')
fraudMarker = mlines.Line2D([], [], linewidth = 0, color='r', marker='.',
                          markersize = 10, label='fraudulent')

plt.legend(handles = [noFraudMarker, fraudMarker], \
           bbox_to_anchor = (1.20, 0.38 ), frameon = False, prop={'size': 16});
```



Fingerprints of genuine and fraudulent transactions

Smoking gun and comprehensive evidence embedded in the dataset of the difference between fraudulent and genuine transactions is obtained by examining their respective correlations in the heatmaps below.

In [26]:
```python
Xfraud = X.loc[Y == 1] # update Xfraud & XnonFraud with cleaned data
XnonFraud = X.loc[Y == 0]

correlationNonFraud = XnonFraud.loc[:, X.columns != 'step'].corr()
mask = np.zeros_like(correlationNonFraud)
indices = np.triu_indices_from(correlationNonFraud)
mask[indices] = True

grid_kws = {"width_ratios": (.9, .9, .05), "wspace": 0.2}
f, (ax1, ax2, cbar_ax) = plt.subplots(1, 3, gridspec_kw=grid_kws, \
                                     figsize = (14, 9))

cmap = sns.diverging_palette(220, 8, as_cmap=True)
ax1 =sns.heatmap(correlationNonFraud, ax = ax1, vmin = -1, vmax = 1, \
    cmap = cmap, square = False, linewidths = 0.5, mask = mask, cbar = False)
ax1.set_xticklabels(ax1.get_xticklabels(), size = 16);
ax1.set_yticklabels(ax1.get_yticklabels(), size = 16);
ax1.set_title('Genuine \n transactions', size = 20)

correlationFraud = Xfraud.loc[:, X.columns != 'step'].corr()
ax2 = sns.heatmap(correlationFraud, vmin = -1, vmax = 1, cmap = cmap, \
 ax = ax2, square = False, linewidths = 0.5, mask = mask, yticklabels = False, \
    cbar_ax = cbar_ax, cbar_kws={'orientation': 'vertical', \
                                 'ticks': [-1, -0.5, 0, 0.5, 1]})
ax2.set_xticklabels(ax2.get_xticklabels(), size = 16);
ax2.set_title('Fraudulent \n transactions', size = 20);

cbar_ax.set_yticklabels(cbar_ax.get_yticklabels(), size = 14);
```



6. Machine Learning to Detect Fraud in Skewed Data

Having obtained evidence from the plots above that the data now contains features that make fraudulent transactions clearly detectable, the remaining obstacle for training a robust ML model is the highly imbalanced nature of the data.

In [27]:
```python
print('skew = {}'.format( len(Xfraud) / float(len(X)) ))
```

```
skew = 0.002964544224336551
```

Split the data into training and test sets in a 80:20 ratio

In [28]:
```python
trainX, testX, trainY, testY = train_test_split(X, Y, test_size = 0.2, \
                                                random_state = randomState)
```

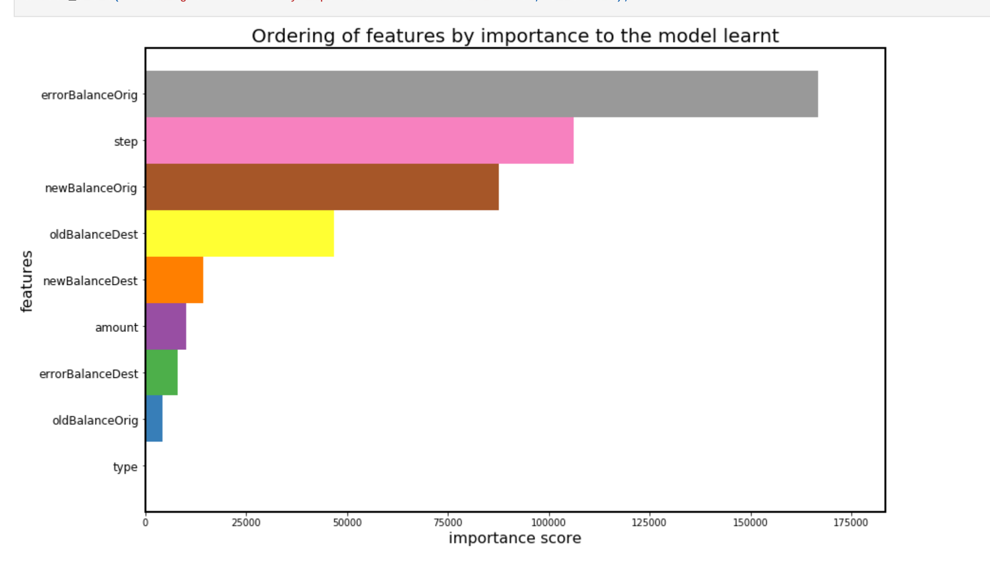
In [29]:
```python
# Long computation in this cell (~1.8 minutes)
weights = (Y == 0).sum() / (1.0 * (Y == 1).sum())
clf = XGBClassifier(max_depth = 3, scale_pos_weight = weights, \
                n_jobs = 4)
probabilities = clf.fit(trainX, trainY).predict_proba(testX)
print('AUPRC = {}'.format(average_precision_score(testY, \
                                              probabilities[:, 1])))
```

```
AUPRC = 0.9986361116985445
```

What are the important features for the ML model?

The figure below shows that the new feature *errorBalanceOrig* that we created is the most relevant feature for the model. The features are ordered based on the number of samples affected by splits on those features.

In [30]:
```python
fig = plt.figure(figsize = (14, 9))
ax = fig.add_subplot(111)

colours = plt.cm.Set1(np.linspace(0, 1, 9))

ax = plot_importance(clf, height = 1, color = colours, grid = False, \
                     show_values = False, importance_type = 'cover', ax = ax);
for axis in ['top','bottom','left','right']:
            ax.spines[axis].set_linewidth(2)

ax.set_xlabel('importance score', size = 16);
ax.set_ylabel('features', size = 16);
ax.set_yticklabels(ax.get_yticklabels(), size = 12);
ax.set_title('Ordering of features by importance to the model learnt', size = 20);
```

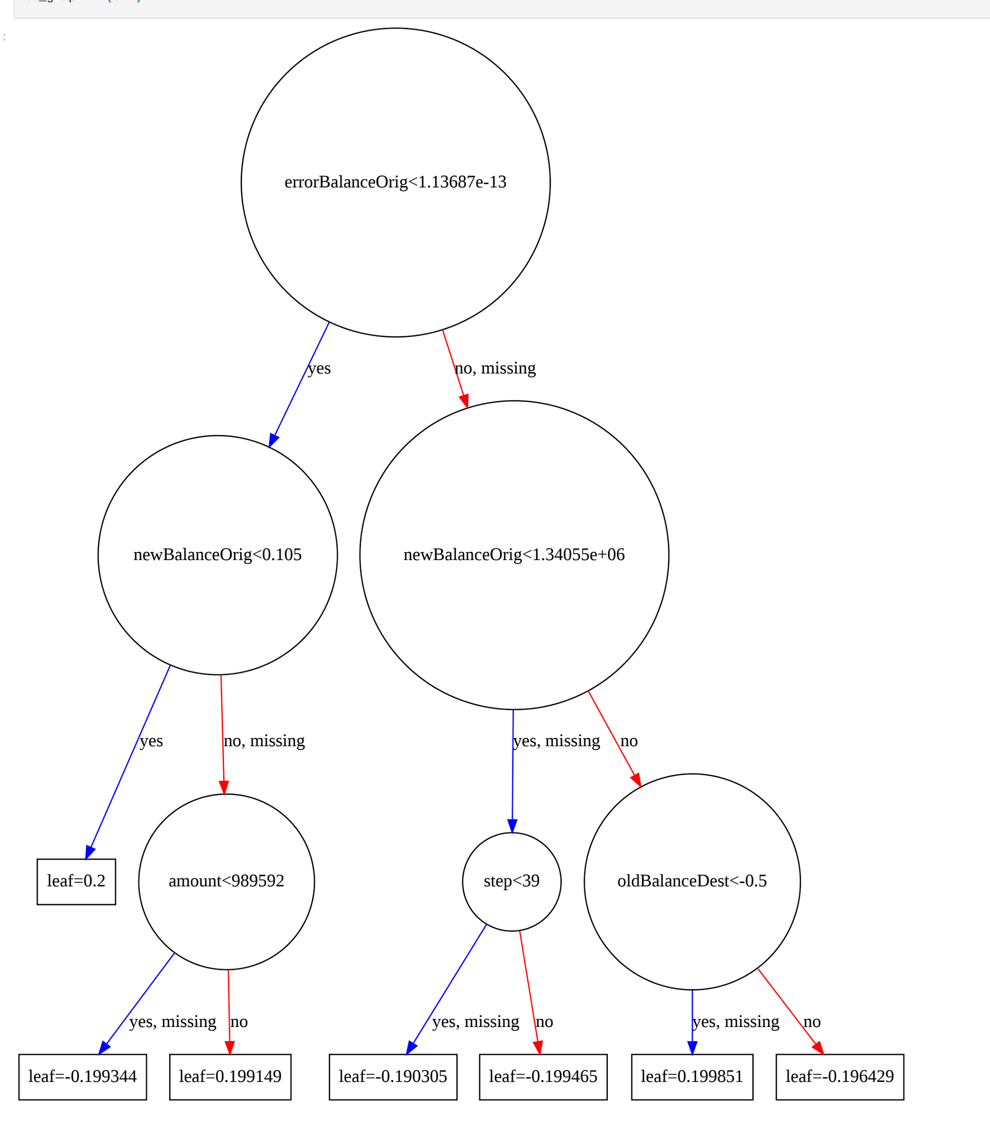


6.2. Visualization of ML model

The root node in the decision tree visualized below is indeed the feature *errorBalanceOrig*, as would be expected from its high significance to the model.

In [31]:
```python
to_graphviz(clf)
```

Out[31]:

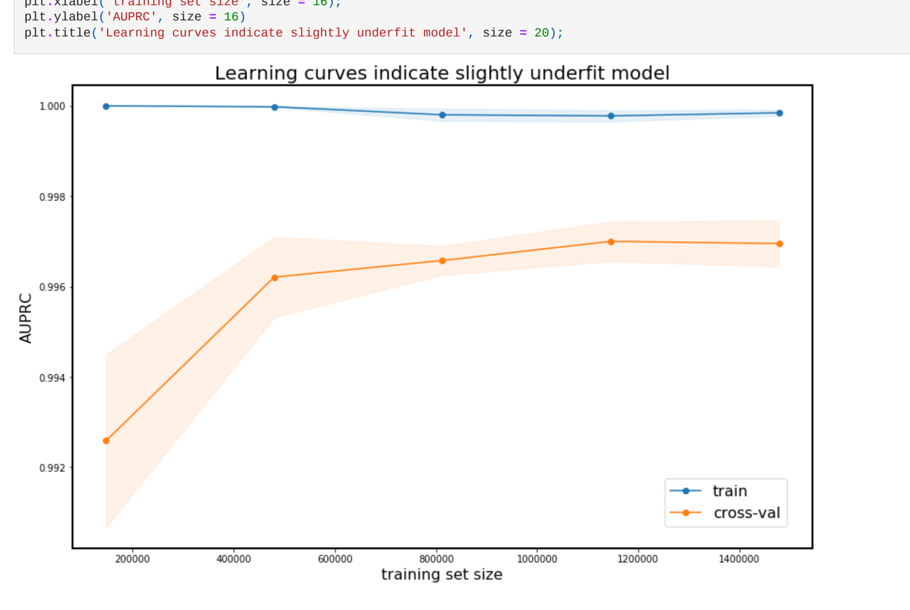


Bias-variance tradeoff

The model we have learnt has a degree of bias and is slighly underfit. This is indicated by the levelling in AUPRC as the size of the training set is increased in the cross-validation curve below. The easiest way to improve the performance of the model still further is to increase the *max_depth* parameter of the XGBClassifier at the expense of the longer time spent learning the model. Other parameters of the classifier that can be adjusted to correct for the effect of the modest underfitting include decreasing *min_child_weight* and decreasing *reg_lambda*.

In [32]:
```python
# Long computation in this cell (~6 minutes)

trainSizes, trainScores, crossValScores = learning_curve(\
XGBClassifier(max_depth = 3, scale_pos_weight = weights, n_jobs = 4), trainX,\
                                         trainY, scoring = 'average_precision')
```

In [33]:
```python
trainScoresMean = np.mean(trainScores, axis=1)
trainScoresStd = np.std(trainScores, axis=1)
crossValScoresMean = np.mean(crossValScores, axis=1)
crossValScoresStd = np.std(crossValScores, axis=1)

colours = plt.cm.tab10(np.linspace(0, 1, 9))

fig = plt.figure(figsize = (14, 9))
plt.fill_between(trainSizes, trainScoresMean - trainScoresStd,
    trainScoresMean + trainScoresStd, alpha=0.1, color=colours[0])
plt.fill_between(trainSizes, crossValScoresMean - crossValScoresStd,
    crossValScoresMean + crossValScoresStd, alpha=0.1, color=colours[1])
plt.plot(trainSizes, trainScores.mean(axis = 1), 'o-', label = 'train', \
         color = colours[0])
plt.plot(trainSizes, crossValScores.mean(axis = 1), 'o-', label = 'cross-val', \
         color = colours[1])

ax = plt.gca()
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)

handles, labels = ax.get_legend_handles_labels()
plt.legend(handles, ['train', 'cross-val'], bbox_to_anchor=(0.8, 0.15), \
               loc=2, borderaxespad=0, fontsize = 16);
plt.xlabel('training set size', size = 16);
plt.ylabel('AUPRC', size = 16)
plt.title('Learning curves indicate slightly underfit model', size = 20);
```



7. Conclusion

We thoroughly interrogated the data at the outset to gain insight into which features could be discarded and those which could be valuably engineered. The plots provided visual confirmation that the data could be indeed be discriminated with the aid of the new features. To deal with the large skew in the data, we chose an appropriate metric and used an ML algorithm based on an ensemble of decision trees which works best with strongly imbalanced classes. The method used in this kernel should therefore be broadly applicable to a range of such problems.