Machine Learning Prediction of Money Laundering Customers

S. J. Curran

Summary

We present a machine learning model which our client, an online gambling platform, can use to detect customers (players) using their infrastructure to launder money. Although we are limited to a sample containing only 1752 suspect players, we report a mean accuracy of 81% in the prediction of a player engaging in suspect activity. The model requires only seven of the 17 available features, potentially increasing its applicability to more players, as well as possibly providing a much larger training sample which could increase the model accuracy.

Contents

1 Introduction

The client, an online gambling platform, requires a machine learning model with which to detect customers (players) who use the client's infrastructure for laundering money through apparent gambling activities. The findings are summarised above, with the following report containing the specific details regarding the building of the model.

2 Analysis

2.1 Pre-processing

2.1.1 The data

The client has supplied five datasets:

1. `details.csv` – the 207 398 players' basic details. The file is known to have formatting issues.
2. `payments.csv` – the payment activities of the players over the last 30 days.
3. `profiling.csv` – additional information regarding the players' activity over the same period.
4. `suspect.csv` – a list of 1752 players already flagged for suspect activity.

The fields are (in order of appearance):

`ID`	the unique ID of the player
`Life_Time`	how many years the player has been a client
`Age`	the client's age in years
`Is_Retail`	whether the player is also a retailer
`Is_CRM_Email`	whether the player has been contacted by us
`CountDeposit`	number of deposits
`CountWithdrawal`	number of withdrawals
`TotalDeposits`	value of deposits
`TotalWithdrawal`	value of withdrawals
`CountPaymentMethod`	number of different deposit/withdrawal methods
`DifferentMethodWithdrawals`	value of different deposit/withdrawal methods
`Multi_Device`	0 flags < 3 devices and 1 flags ≥ 3 devices used for payment
`IP_Counts`	number of different IP addresses used for transactions
`e_11234, e_23456, e_34454 e_43568, e_64645`	aggregated variables describing the players' activity

2.1.2 First look at the data

Using `pre-process.py`, we see that `details.csv` does indeed have formatting issues:

```
                                ID|Life_Time|Age|Is_Retail|Is_CRM_Email
NaN db=123 | 25 | 100001096=id     | | 1 )| | | 33 | | )| 1 | | | 0 |
    db=123 | 25 | 100002057=id     | | 1 )| | | 44 | | )| 0 | | | 0 |
 ...                                                                ...
    db=123 | 25 | 899994603=id     | | 3 )| | | 35 | | )| 0 | | | 0 |
    db=123 | 25 | 899998254=id     | | 0 )| | | 39 | | )| 0 | | | 0 |
[207398 rows x 1 columns]
```

This was easily fixed in the emacs editor[1] and saved as `details_fixed.csv`

```
               ID  Life_Time  Age  Is_Retail  Is_CRM_Email
0       100001096          1   33          1             0
1       100002057          1   44          0             0
...           ...        ...  ...        ...           ...
207396  899994603          3   35          0             0
207397  899998254          0   39          0             0

[207398 rows x 5 columns]
                 ID      Life_Time            Age      Is_Retail   Is_CRM_Email
count  2.073980e+05  207398.000000  207398.000000  207398.000000  207398.000000
mean   5.008909e+08       2.603342      41.490410       0.410023       0.418042
std    2.306288e+08       3.122641      13.123745       0.491839       0.493238
min    1.000011e+08       0.000000      18.000000       0.000000       0.000000
max    8.999983e+08      21.000000     120.000000       1.000000       1.000000
```

Looking at `payments.csv`,

```
               ID  CountDeposit  ...  CountPaymentMethod  DifferentMethodWithdrawals
0       100001096            86  ...                   1                      2359.8
1       100002057            26  ...                   1                         0.0
...           ...           ...  ...                 ...                         ...

207396  899994603             1  ...                   1                         0.0
207397  899998254             5  ...                   1                         0.0
```

this has the customer ID (`ID`) in common with `details.csv`, as does `profiling.csv`.

[1] Command line methods such as sed, awk, etc. could also have been used.

	ID	Multi_Device	IP_Counts	e_11234	e_23456	e_34454	e_43568	e_64645
0	100001096	1	3	5599.89	986.36	283.05	662.98	0.40
1	100002057	1	1	9669.93	17181.08	20905.39	38635.13	3.22
...	...	...	...	...	...	...	...	...
207396	899994603	1	2	7522.49	16485.09	37475.16	37475.16	3.15
207397	899998254	1	1	9229.69	3589.55	578.35	31473.88	1.22

We therefore merged these datasets by `ID` and checking the merged data we find no missing values.

	Life_Time	Age	...	e_34454	e_43568	e_64645
count	207398.000000	207398.000000	...	207398.000000	207398.000000	207398.000000
mean	2.603342	41.490410	...	14666.698999	18722.672481	1.516427
std	3.122641	13.123745	...	13152.810961	15426.545697	1.101400
min	0.000000	18.000000	...	0.000000	0.000000	0.000000
25%	0.000000	31.000000	...	3454.317500	3320.920000	0.490000
50%	1.000000	39.000000	...	10207.420000	14811.630000	1.350000
75%	4.000000	51.000000	...	25097.677500	34063.392500	2.670000
max	21.000000	120.000000	...	70086.760000	56891.910000	4.310000

[8 rows x 18 columns]

Finally, looking at `suspect.csv`, this only contains only two fields, the player ID and a positive flag (`Target_ml` = 1) for suspected laundering activity.

	ID	Target_ml
count	1.752000e+03	1752.0
mean	4.995049e+08	1.0
std	2.322098e+08	0.0
min	1.000955e+08	1.0
max	8.997882e+08	1.0

All 1752 of these players were also in the merged file, which was absent a suspect activity flag. These flags were added to the merged file and, assuming that the $207\,389 - 1752 = 205\,637$ remaining players are non-suspect, we flagged these as `Target_ml` = 0.

	ID	Life_Time	Age	...	e_23456	e_34454	e_43568	e_64645	Target_ml
0	100001096	1	33	...	986.36	283.05	662.98	0.40	0
1	100002057	1	44	...	17181.08	20905.39	38635.13	3.22	0
...	...	...	...	...	...	...	...	...	...
207396	899994603	3	35	...	16485.09	37475.16	37475.16	3.15	0
207397	899998254	0	39	...	3589.55	578.35	31473.88	1.22	0

[207398 rows x 19 columns]

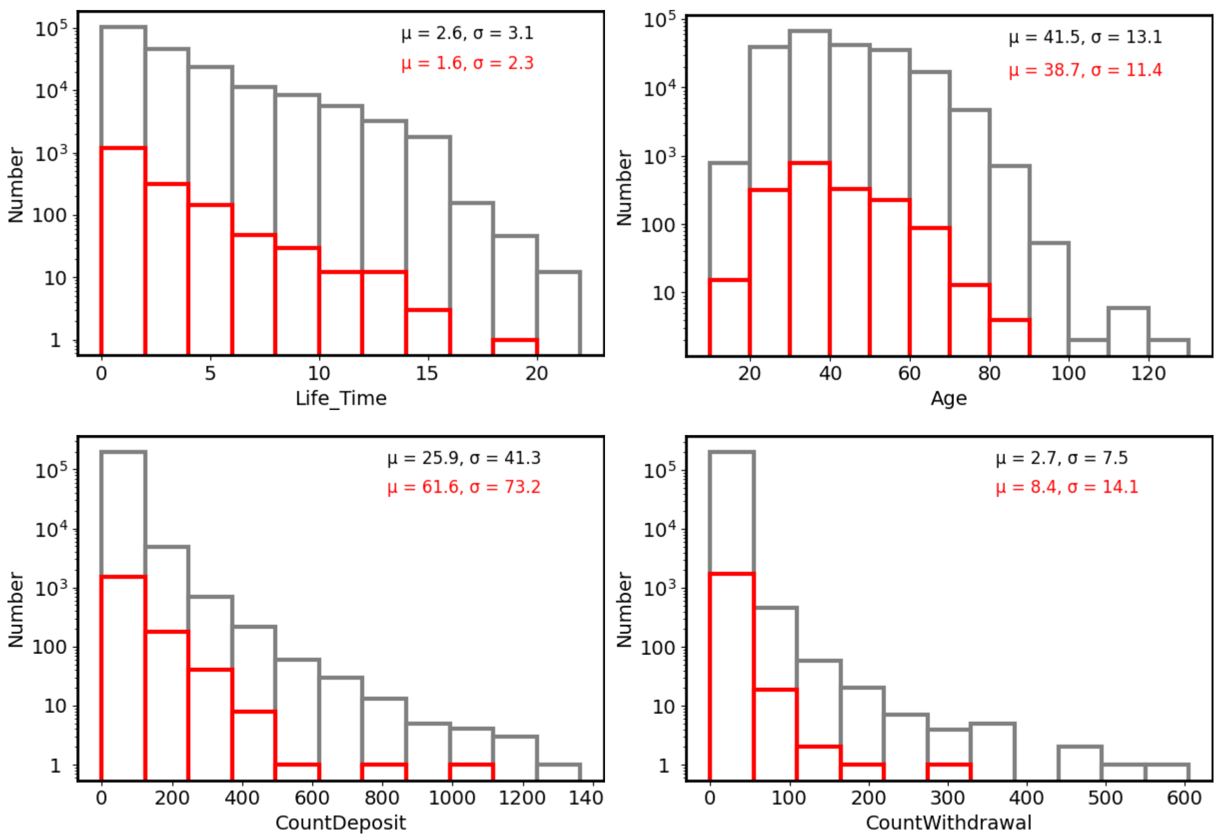
	Life_Time	Age	...	e_43568	e_64645	Target_ml

count	207398.000000	207398.000000	...	207398.000000	207398.000000	207398.000000
mean	2.603342	41.490410	...	18722.672481	1.516427	0.008448
std	3.122641	13.123745	...	15426.545697	1.101400	0.091522
min	0.000000	18.000000	...	0.000000	0.000000	0.000000
max	21.000000	120.000000	...	56891.910000	4.310000	1.000000

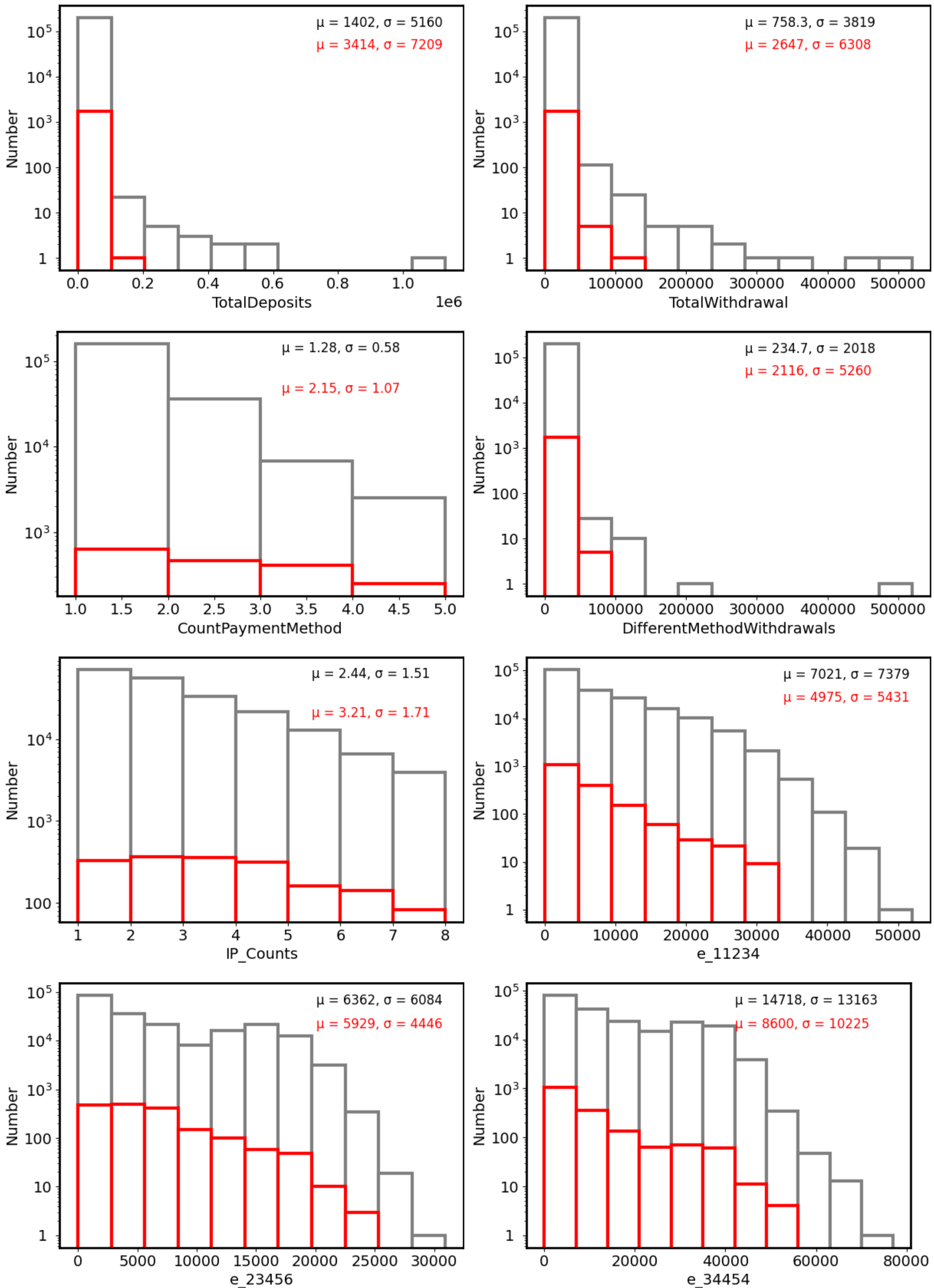
We then save these data to `all_data.csv` to be used in the machine learning.

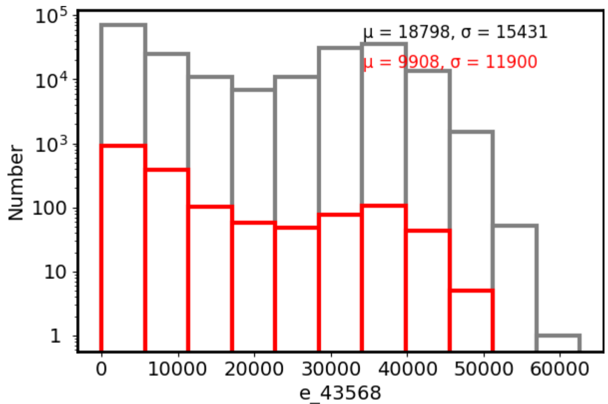
2.1.3 Distributions

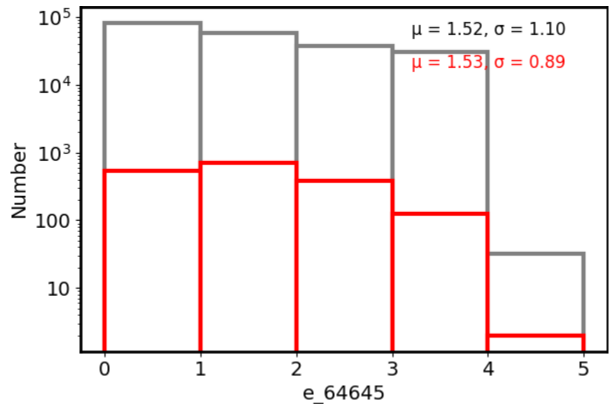
In order to eyeball any obvious differences between the suspect and non-suspect players, we plot the distributions of the non-binary features.



From these we see that the suspect players features (red histograms) mostly overlap with the non-suspect plays (black histograms), demonstrating that these would be difficult to distinguish from one another based the distributions alone. Note also that some of the non-suspect players are suspiciously long-lived.


μ = 1402, σ = 5160
μ = 3414, σ = 7209
Number
TotalDeposits
1e6
μ = 758.3, σ = 3819
μ = 2647, σ = 6308
Number
TotalWithdrawal
μ = 1.28, σ = 0.58
μ = 2.15, σ = 1.07
Number
CountPaymentMethod
μ = 234.7, σ = 2018
μ = 2116, σ = 5260
Number
DifferentMethodWithdrawals
μ = 2.44, σ = 1.51
μ = 3.21, σ = 1.71
Number
IP_Counts
μ = 7021, σ = 7379
μ = 4975, σ = 5431
Number
e_11234
μ = 6362, σ = 6084
μ = 5929, σ = 4446
Number
e_23456
μ = 14718, σ = 13163
μ = 8600, σ = 10225
Number
e_34454





2.2 Machine Learning

2.2.1 Initial models and tests

In `ML.py` we initially test four types of common, but different, ML classifiers:

1. *Logistic Regression* (LR): This is analogous to multi-variable linear regression, but instead of a fit yields a binary result. Thus, it is particularly suited to this problem where we are classifying a player as suspect or not. The algorithm compresses a linear combination of several variables (features) with a logistics sigmoid to yield a value of between 0 and 1. For the binary model, the prediction is labelled with one of these two end values, depending upon its probability $(0 \rightarrow 1)$, or odds $(0 \rightarrow \infty)$

2. *k-Nearest Neighbour* (kNN): This algorithm maps the variables to a feature space and then compares the Euclidean distance between a test point and its k nearest neighbours. It then assigns a weighted combination of the target values with the nearest neighbours in order to place the test object in a group. The kNN algorithm is relatively computationally expensive and, like logistic regression, is sensitive to outliers. A further disadvantage is that irrelevant features can lead the learning astray.

3. *Support Vector Classifier* (SVC): This constructs a hyperplane in a high dimensional space in order to perform classification, regression and outlier detection. Support vectors are points that reside closest to the hyperplane and in binary classification the training maximises the distance between the two categories. Further data are transformed into the same space and assigned a category based upon where in the space the are located. Although the support vector machine classifier is computationally fast, it is not suitable for noisy data (overlapping features) nor large data sets.

4. *Decision Tree Classifier* (DTC): Like the other algorithms, decision trees can be used for both classification and regression. The algorithm builds a classification model based upon a tree structure, which branches the data-set (top node) into smaller subsets (child nodes), according to a predefined decision boundary. With one node on either side of the boundary, the process is iterated through further branching until a predefined stopping criterion in reached. DTC has the advantage that it is not as sensitive to outliers as some of the other algorithms, although the tree choices can be biased

due to the sequential nature of the algorithm. Over-fitting can also be a problem, which can be mitigated by limiting the maximum tree depth.

As stated above, there are only 1752 suspect players and so from raw statistics alone we can ascertain that there is a probability of $100 \times 1752/207\,389 = 0.84\%$ of the player being suspect. In order to prevent the ML blindly yielding an apparent $100 - 0.84 \approx 99\%$ accuracy from probability alone, we randomly selected 1752 of the remaining 205 637 non-suspect players for the binary testing.

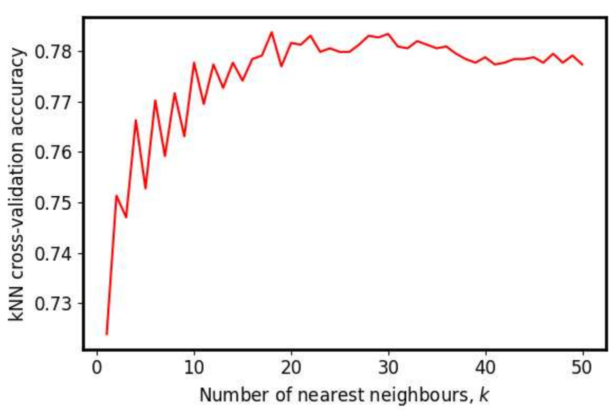
Training on 80% of the 2×1752 strong sample (2803 players) and testing on the other 20% (701), using the default values of the algorithms, we obtain the following cross-validation scores (in percent).

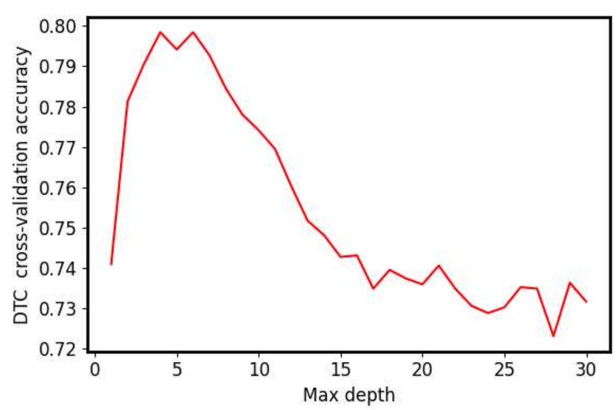
```
For a 0.2 test fraction (2803 train & 701 test) LR score = 76.597
For a 0.2 test fraction (2803 train & 701 test) KNN score = 78.880
For a 0.2 test fraction (2803 train & 701 test) SVC score = 81.662
For a 0.2 test fraction (2803 train & 701 test) DTC score = 80.448
```

2.2.2 Model optimisation

We then optimise each of the models via:

- *Logistic regression* – using `sklearn GridSearchCV` giving `LR.py`[2], `C=0.004833`, `solver='newton-cg'`.
- *Support Vector Classifier* – using `sklearn GridSearchCV` giving `SVC.py`, `C=1, gamma=0.1`.
- *k-Nearest Neighbour* & *Decision Tree Classifier* – we iterated the number of nearest neighbours and maximum depths, giving `n_neighbors` = 26 and `max_depth` = 6, respectively.





Re-running `ML.py` with these parameters we obtain

```
For a 0.2 test fraction (2803 train & 701 test) LR score = 76.026
For a 0.2 test fraction (2803 train & 701 test) KNN score = 79.987
```

[2] Warning, this takes a while!

```
For a 0.2 test fraction (2803 train & 701 test) SVC score = 81.485
For a 0.2 test fraction (2803 train & 701 test) DTC score = 80.450
```

indicating that the "default" values were close to optimal and that SVC is the best model.

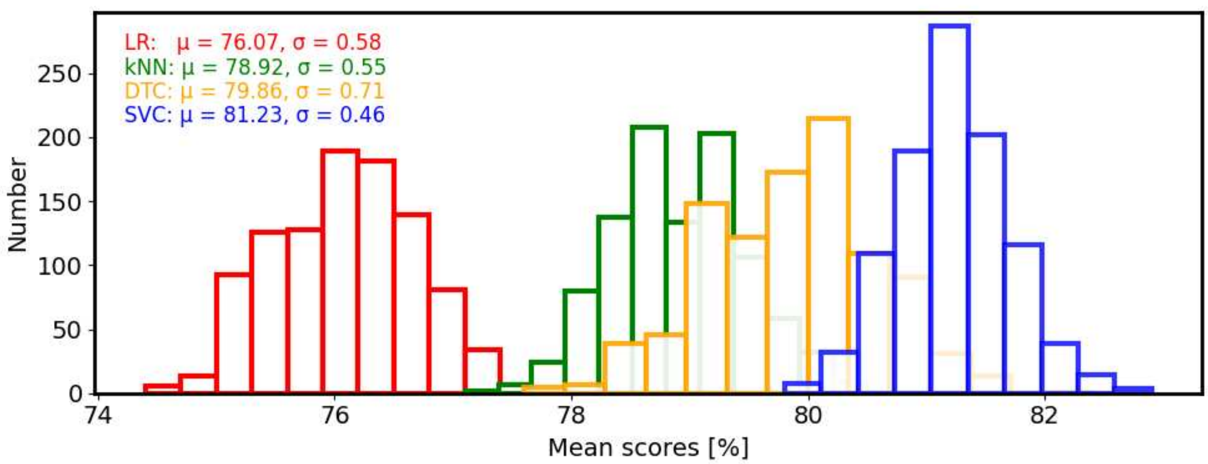
Below we show the confusion matrix for the run of the SVC algorithm.

	Predicted training		**Predicted validation**	
	Suspect	**Non-suspect**	**Suspect**	**Non-suspect**
Actual suspect	1372	410	321	109
Actual non-suspect	42	979	17	254

The validation data yield 321 true positives (suspect) and 254 true negatives (non-suspect), compared to 17 false positives and 109 false negatives. This gives a test score of $(321 + 254)/(321 + 254 + 17 + 109) = 0.82$, the cross-validation score.

However, as above, these tests are based upon a single instance of the models and, given the small sample, could be subject to the choice of the 1752 non-suspect players. Using `ML-loop.py` we run 1000 trials randomising the sample of non-suspect players from the pool of 205 646 each time.[3]

Summarising the results in the histograms below, we see that the SVC algorithm performs best with a mean cross-validation score of $81.23 \pm 0.46\%$.



2.2.3 Feature importance

Below, from `ML.py`, we show the feature importance for each algorithm in descending order from a single run.

```
LogisiticRegression score = 76.524
              Feature  Importance
   CountPaymentMethod    0.053086
              e_64645    0.024331
```

```
Support Vector Classifier score = 81.55
              Feature  Importance
              e_23456    0.067499
              e_64645    0.055369
```

[3]From 207 398 total minus the 1752 suspect players.

e_43568	0.022975
IP_Counts	0.016625
e_23456	0.010275
e_34454	0.009989
Is_Retail	0.006136
DifferentMethodWithdrawals	0.005066
Multi_Device	0.003568
Life_Time	0.003211
CountDeposit	0.001570
CountWithdrawal	0.001284
e_11234	0.000357
Is_CRM_Email	0.000214
Age	0.000143
TotalDeposits	-0.000571
TotalWithdrawal	-0.001070

KNearest score = 78.843

Feature	Importance
e_64645	0.043453
e_23456	0.038673
e_43568	0.035391
CountPaymentMethod	0.026543
Is_Retail	0.026258
e_11234	0.016054
e_34454	0.014413
IP_Counts	0.012558
Is_CRM_Email	0.009704
Age	0.009633
Multi_Device	0.008348
Life_Time	0.008063
CountDeposit	0.005280
CountWithdrawal	0.003068
TotalDeposits	0.002426
TotalWithdrawal	0.001927
DifferentMethodWithdrawals	0.001213

e_43568	0.051445
IP_Counts	0.017838
Is_Retail	0.017339
e_34454	0.016768
CountPaymentMethod	0.015912
DifferentMethodWithdrawals	0.012986
e_11234	0.012772
Life_Time	0.011559
TotalDeposits	0.010917
CountDeposit	0.008919
TotalWithdrawal	0.008205
CountWithdrawal	0.006778
Multi_Device	0.006422
Is_CRM_Email	0.005351
Age	0.005351

DecisionTreeClassifier score = 78.701

Feature	Importance
CountWithdrawal	0.231038
TotalDeposits	0.140564
DifferentMethodWithdrawals	0.070781
CountDeposit	0.068783
e_23456	0.048519
e_64645	0.040314
IP_Counts	0.022333
e_43568	0.020407
TotalWithdrawal	0.018480
e_34454	0.012130
Life_Time	0.010560
e_11234	0.001998
Age	0.000999
Is_CRM_Email	0.000999
Multi_Device	0.000000
Is_Retail	0.000000
CountPaymentMethod	0.000000

These were found to be roughly consistent, although there was some variation between different runs. Focusing on the best performing (SVC) algorithm, we found that the score remained above 80% if we retained only the top seven features. Therefore, we test each classifier with only the seven most important features listed for each one above.

The results are summarised in the following table, where we see the top features are similar between the LR, SVC and kNN classifiers, with only `e_64645` being common to all. Bear in mind, however, this is based on the single run shown above. Again, running 1000 trials, where we run *all* top seven classifier features for each one (`ML-loop_feat.py`), we see that the results are largely insensitive to the choice of the seven features, suggesting that the most important features are a subset of those above.

Feature	Top Seven Classifier Features				Number of
	LR	SVC	kNN	DTC	Occurrences
e_11234			✓		1
e_23456	✓	✓	✓		3
e_34454	✓	✓	✓		3
e_43568	✓	✓	✓		3
e_64645	✓	✓	✓	✓	4
CountDeposit				✓	1
CountPaymentMethod	✓	✓	✓		3
CountWithdrawal				✓	1
DifferentMethodWithdrawals	✓			✓	2
Is_Retail	✓	✓	✓		3
IP_Counts		✓		✓	2
TotalDeposits				✓	1
Classifier mean score [%]					**All features**
LR	74.95 ± 0.62	74.92 ± 0.62	74.93 ± 0.61	72.01 ± 0.70	76.07 ± 0.58
SVC	80.34 ± 0.46	80.33 ± 0.45	80.47 ± 0.45	79.51 ± 0.47	81.23 ± 0.46
kNN	80.44 ± 0.52	80.41 ± 0.52	81.04 ± 0.47	78.60 ± 0.57	78.92 ± 0.55
DTC	78.68 ± 0.68	78.71 ± 0.65	78.55 ± 0.65	79.72 ± 0.64	79.86 ± 0.71

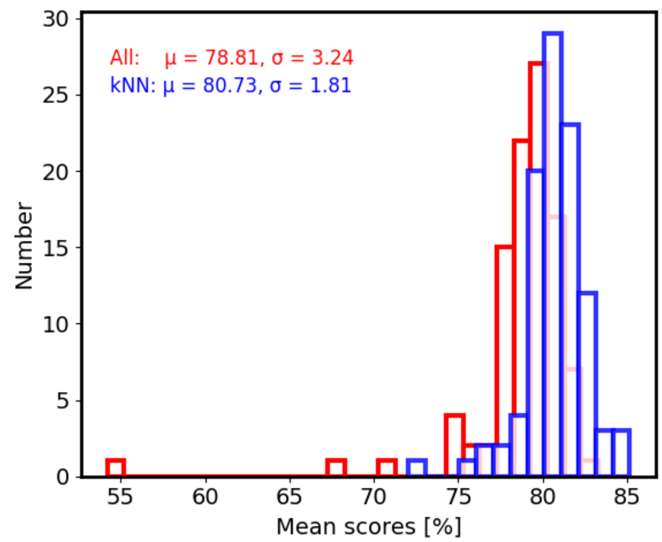
We also see that the LR classifier remains the worst performing for these data, in addition to a marked improvement in the kNN performance. In fact, from the highlighted values, we see that this is the best performer for the LR, SVC and kNN features. As was noted in Sect. 2.2.1, the effectiveness of the kNN algorithm is susceptible to irrelevant features which may be what we are seeing here.

2.3 Deep Learning

Although, due to having only 1752 suspect players available for testing, the sample is likely too small to yield reliable predictions from a deep learning algorithm, we nevertheless test this in case any further insight can be gained.

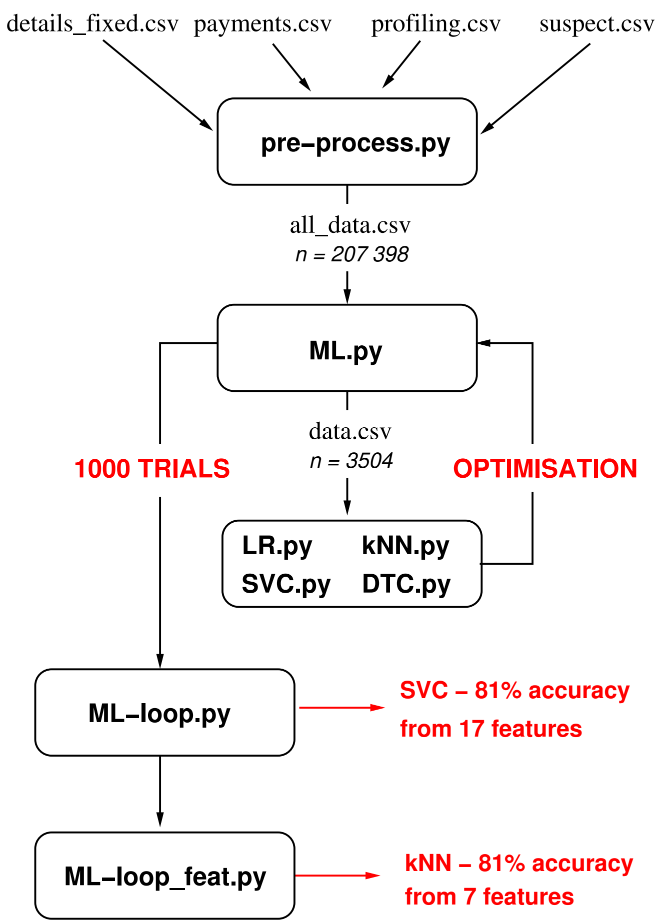
In `DL.py` we use TensorFlow, with the testing of various hyper-parameters giving the best results with two *ReLu* layers comprising ≈ 50 neurons each.

In order to get an accurate representation of the results, in `DL-loop.py` we run the deep learning algorithm 100 times using all of features as well as the top seven for the kNN classifications (see above). From this we see that, although the means of the scores are similar to the machine learning models, the spreads are significantly wider, meaning that relying on this deep learning model carries a significant risk of being much less accurate than the machine learning.



3 Discussion

We summarise the steps taken in the flow diagram below.

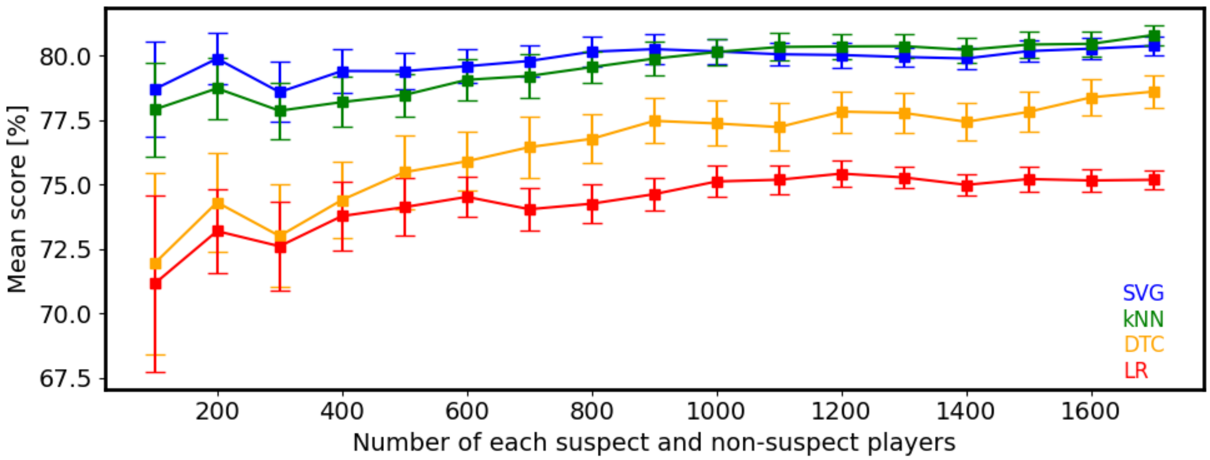


- In `pre-process.py` we combine the players details (`details_fixed.csv`) with their payment activities (`payments.csv`) and other additional information (`profiling.csv`). Using `suspect.csv` we then flag those with known suspect activity.
- In `ML.py` we select the 1752 suspect players in addition to a random sample of 1752 non-suspect. Four different classification algorithms are tested on the sample of 3504 and then optimised.
- Selecting such a relatively small number of non-suspects from a pool of 205 646 may lead to a single trial being unrepresentative. Therefore, in `ML-loop.py` we randomise the selection of the 1752 non-suspect players and re-run the algorithm. After 1000 runs, we find the Support Vector Classifier to be the best performer with a mean score of $81.23 \pm 0.46\%$.
- Finally, we look at the relative importance of each feature and find that retaining only the seven most

important has little detrimental effect. In the case of the kNN classifier the performance reaches the same level as the full 17 feature SVC classifier.

Note that the requirement of only seven features has the potential to vastly increase the sample sizes and number of players which can be tested for likely money laundering.[4]

Lastly, we note that the sample is small, being trained on just 2×1752 players. Increasing the sample size is expected to increase the accuracy and, in order to investigate this, using `ML_sampled.py`, we show the mean scores from 100 trials for each of the algorithms, with the top seven kNN features for randomly selected samples of varying size.[5]



This suggests that the accuracy does not increase significantly for sample sizes above 1000 suspect and 1000 non-suspect players, although the spread in the scores narrows and the DTC classifier is trending strongly upwards. In any case, a significant increase in accuracy cannot be ruled out until much larger datasets are used for the training.

[4]The SVC and kNN models, which use the top seven kNN features, have been saved in `KNN-0.818.pickle` and `SVC-0.812.pickle`, which as the names suggest, have training scores of 81.8% and 81.2%, respectively.

[5]Plotted with `sampled_plot.py`