Machine Learning Prediction of Money Laundering Customers

S. J. Curran

Overview

I present a machine learning model which the client, an online gambling platform, can use to detect customers (players) using their infrastructure to launder money. Although limited to a sample containing only 1752 suspect players, a mean accuracy of $\approx 89\%$ in the prediction of a player engaging in suspect activity is attained. 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.

Although all of the processing was done in the python scripts mentioned in the report, a simplified run-through is included as the Jupyter notebook `PP+ML.ipynb`

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`, it is seen 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

These datasets were therefore merged, by `ID`, and checking the merged data no missing values are found.

	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 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, these were flagged 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]

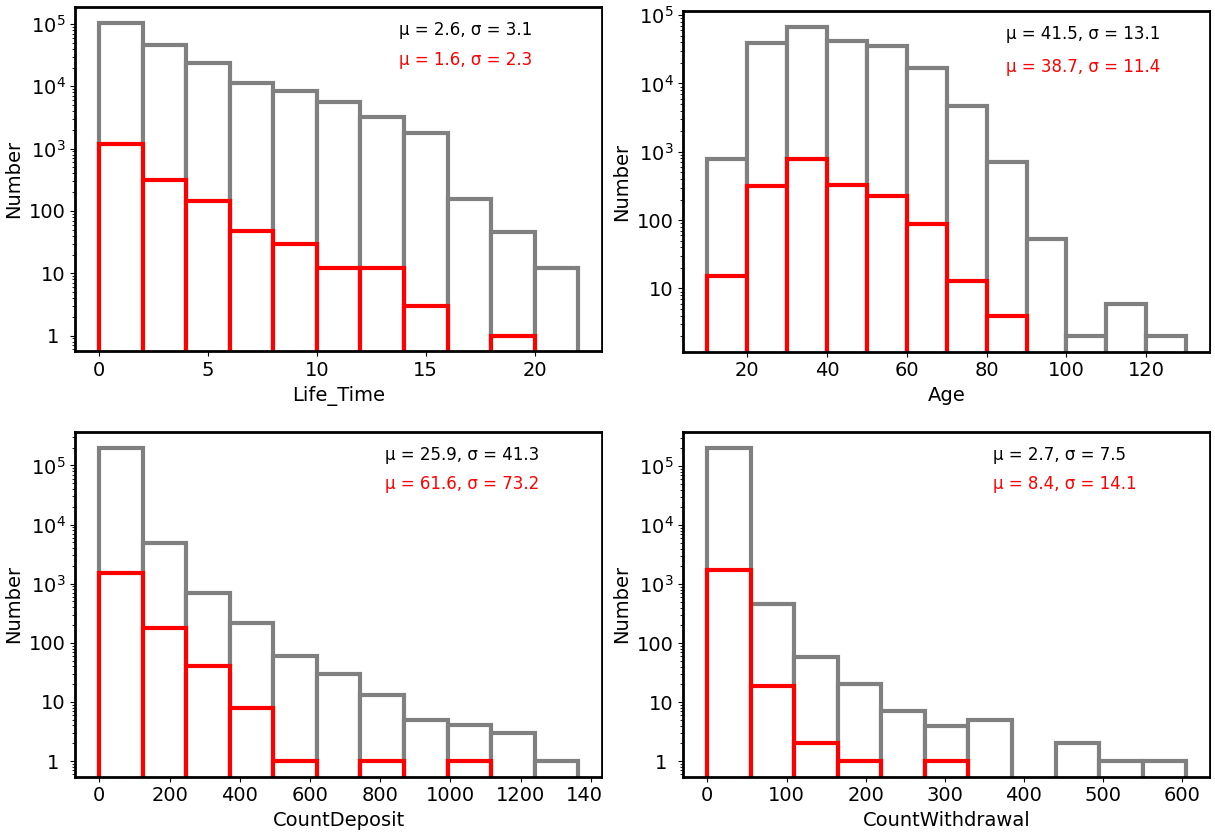
	Life_Time	Age	...	e_43568	e_64645	Target_ml

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

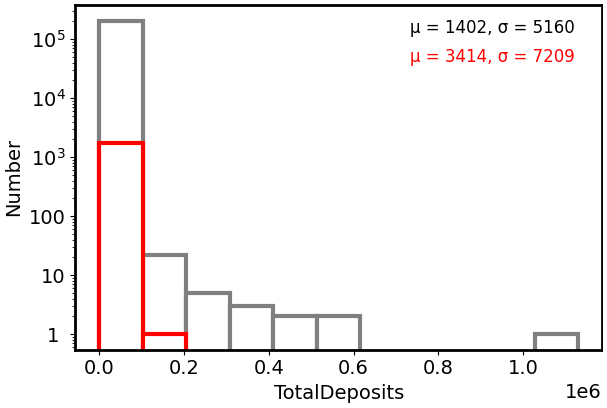
The data were then saved to `all_data.csv` to be used in the machine learning.

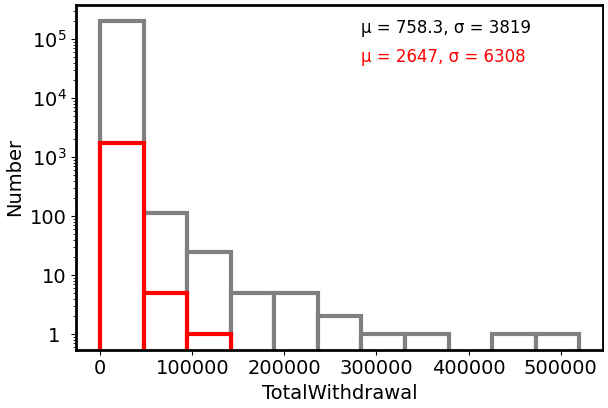
2.1.3 Distributions

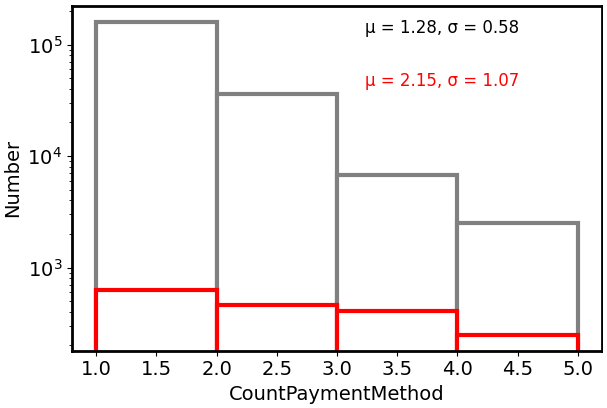
In order to visualise the data and eyeball any obvious differences between the suspect and non-suspect players, the distributions of the non-binary features are plotted below.

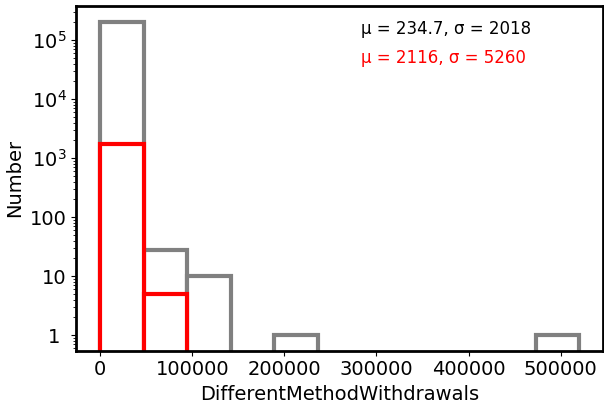


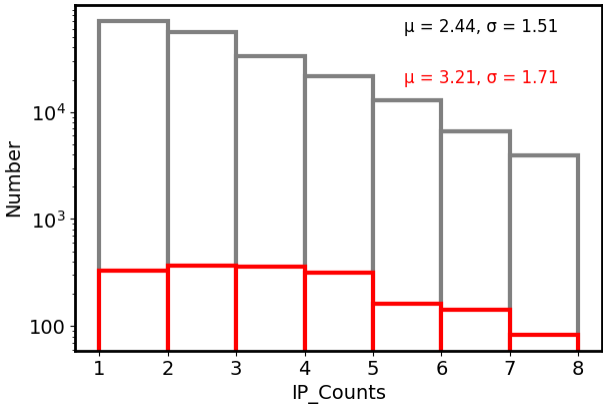
It is seen that the suspect players features (red histograms) mostly overlap with the non-suspect plays (black histograms), demonstrating that these would be difficult to distinguish from one another based the distributions alone. Note also that some of the non-suspect players are suspiciously long-lived.

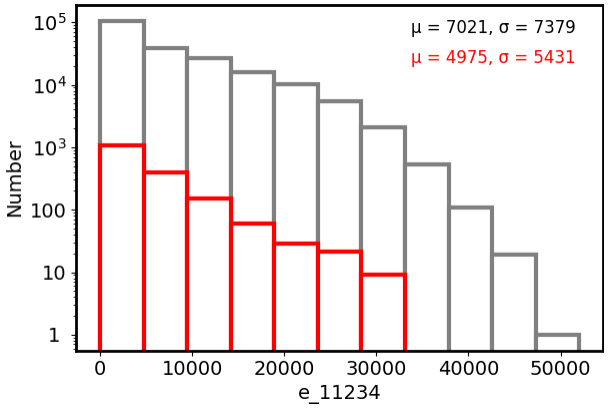

μ = 1402, σ = 5160
μ = 3414, σ = 7209
Number
TotalDeposits
1e6

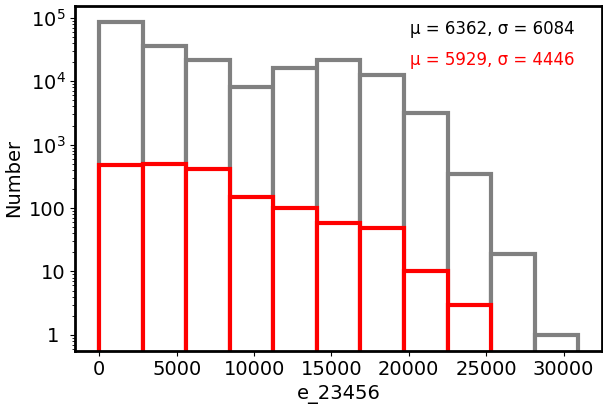

μ = 758.3, σ = 3819
μ = 2647, σ = 6308
Number
TotalWithdrawal

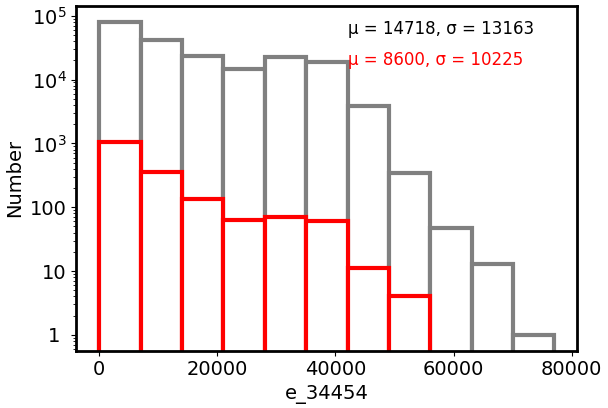

μ = 1.28, σ = 0.58
μ = 2.15, σ = 1.07
Number
CountPaymentMethod

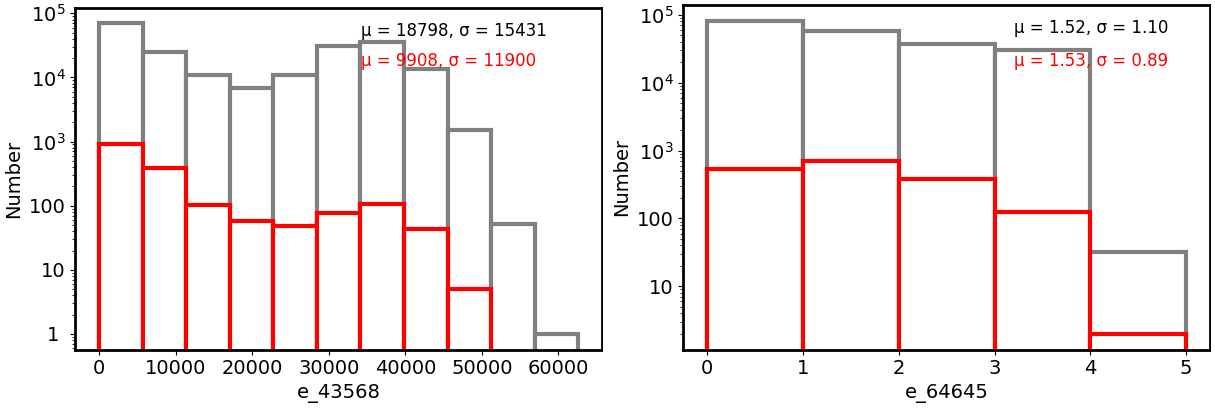

μ = 234.7, σ = 2018
μ = 2116, σ = 5260
Number
DifferentMethodWithdrawals


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` four types of common, but different, ML classifiers were initially tested:

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.

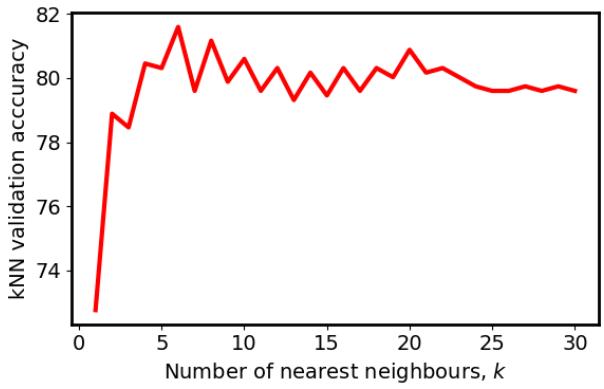
As stated above, there are only 1752 suspect players and so from raw statistics alone we can ascertain that there is a probability of $100 \times 1752/207\,389 = 0.84\%$ of the player being suspect. In order to prevent the ML blindly yielding an apparent $100 - 0.84 \approx 99\%$ accuracy based upon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

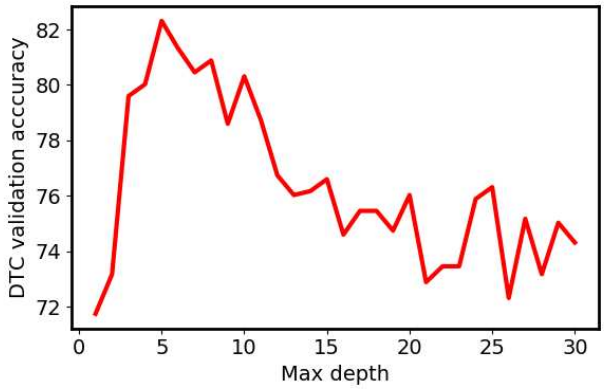
Training on 80% of the 2×1752 strong sample (2803 players) and testing on the other 20% (701), using the default values of the algorithms,training scores of $\gtrsim 75\%$ and validation scores $\gtrsim 70\%$ were obtained.

2.2.2 Model optimisation

Each models was then optimised via:

- *Logistic regression* – using `sklearn GridSearchCV` in `LR.py`, giving `C=0.0886`, `solver='newton-cg'`.
- *Support Vector Classifier* – using `sklearn GridSearchCV` in `SVC.py`, giving `C=1, gamma=0.1`.
- *k-Nearest Neighbour & Decision Tree Classifier* – the number of nearest neighbours and maximum depths were iterated (in `kNN.py` & `DTC.py`), giving `n_neighbors` = 6 and `max_depth` = 5, respectively.





Re-running `ML.py`, with these parameters the results were similar to before.

```
For a 0.2 test fraction (2803 train & 701 test) LR training score = 75.135 percent
Validation accuracy of LR is 78.60 percent

For a 0.2 test fraction (2803 train & 701 test) KNN training score = 78.131 percent
Validation accuracy of KNN is 80.17 percent

For a 0.2 test fraction (2803 train & 701 test) SVC training score = 80.093 percent
```

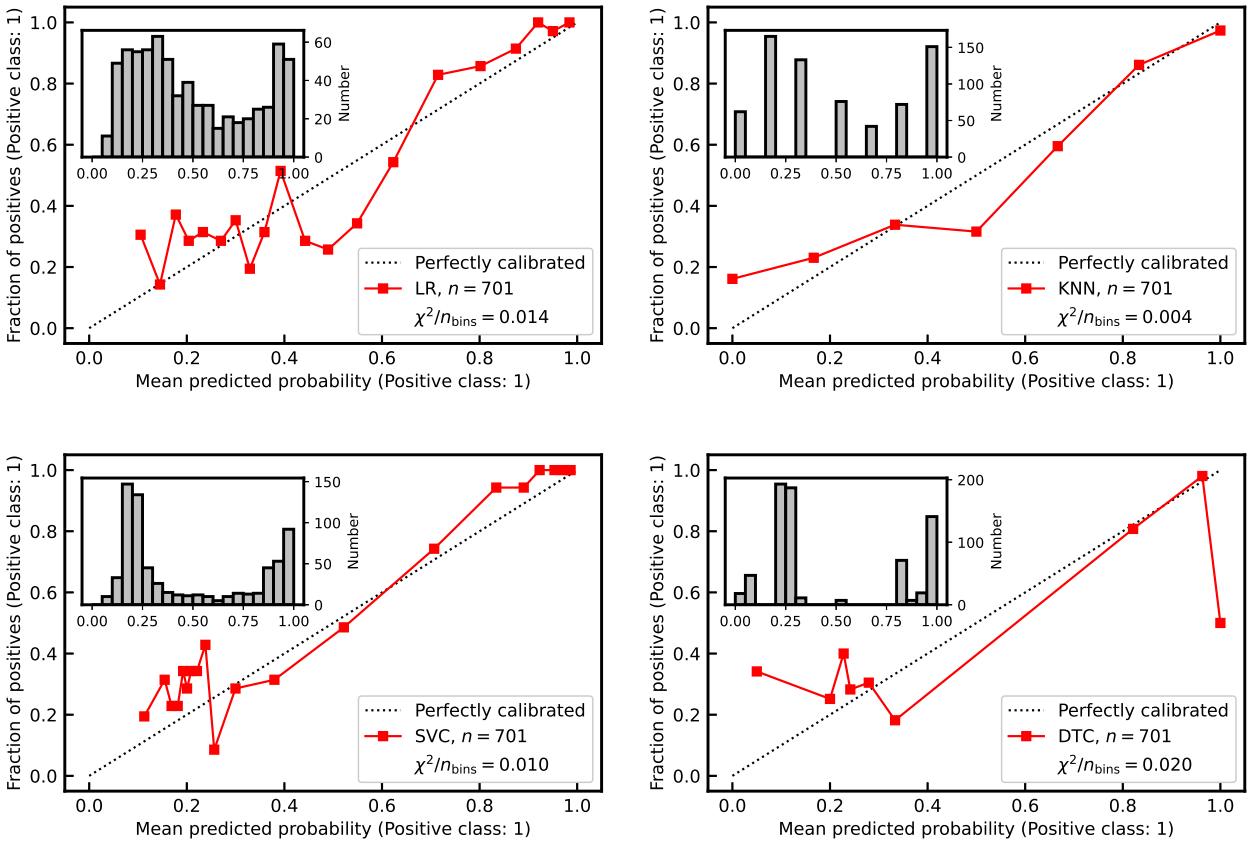
```
Validation accuracy of SVC is 82.17 percent

For a 0.2 test fraction (2803 train & 701 test) DTR training score = 78.879 percent
Validation accuracy of DTR is 77.89 percent
```

The confusion matrix for a run of the SV classifier is shown to the right. From the validation of the model, TP is the number of true positives, FP – false positives, FN – false negatives and TN – true negatives.

	Predicted			
	Good	**Bad**		
Actual good	323	103	TP	FP
Actual bad	22	253	FN	TN

The validation data yield 323 true positives (suspect) and 253 true negatives (non-suspect), compared to 22 false positives and 103 false negatives. This gives a test score of (323 + 253)/(323 + 253 +22 + 103) = 0.8217.

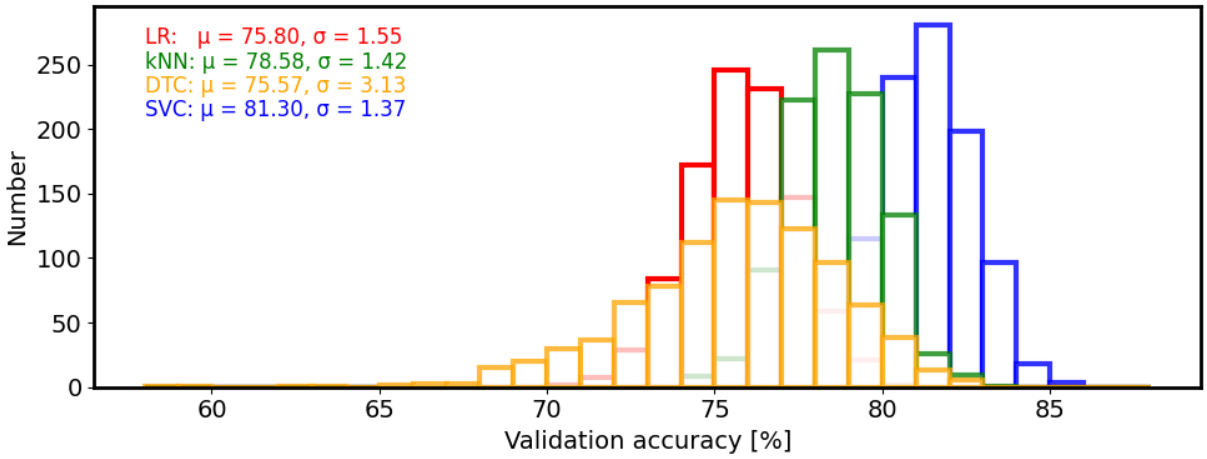


However, from the calibration plots it appears that the kNN classification may be the best, with the least straying from perfect calibration ($\chi^2/n_{\text{bins}} = 0.004$ in this instance). This is also reflected in the histogram, which is fairly even distributed, whereas the SV classifier tends to push extreme probabilities (≈ 0.2 or ≈ 1).

Furthermore, 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`, this is tested by running 1000 trials, randomising the sample of non-suspect players from the pool of 205 646 each time.[2]

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

Summarising the results in the histograms below, averaging a validation score of $81.3 \pm 1.4\%$ it is seen that the SV classifier outperforms the others. The kNN classifier comes in at a close second and, given the calibration plots above, may be the preferred model.



2.2.3 Feature importance

In descending order, from a single run the importance of the features are:

LogisiticRegression score = 76.524

Feature	Importance
CountPaymentMethod	0.053086
e_64645	0.024331
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

Support Vector Classifier score = 81.55

Feature	Importance
e_23456	0.067499
e_64645	0.055369
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

KNearest score = 78.843

Feature	Importance

DecisionTreeClassifier score = 78.701

Feature	Importance

e_64645	0.043453	CountWithdrawal	0.231038
e_23456	0.038673	TotalDeposits	0.140564
e_43568	0.035391	DifferentMethodWithdrawals	0.070781
CountPaymentMethod	0.026543	CountDeposit	0.068783
Is_Retail	0.026258	e_23456	0.048519
e_11234	0.016054	e_64645	0.040314
e_34454	0.014413	IP_Counts	0.022333
IP_Counts	0.012558	e_43568	0.020407
Is_CRM_Email	0.009704	TotalWithdrawal	0.018480
Age	0.009633	e_34454	0.012130
Multi_Device	0.008348	Life_Time	0.010560
Life_Time	0.008063	e_11234	0.001998
CountDeposit	0.005280	Age	0.000999
CountWithdrawal	0.003068	Is_CRM_Email	0.000999
TotalDeposits	0.002426	Multi_Device	0.000000
TotalWithdrawal	0.001927	Is_Retail	0.000000
DifferentMethodWithdrawals	0.001213	CountPaymentMethod	0.000000

These were found to be roughly consistent, although there was some variation between different runs. Focusing on the best performing (SV) classifier, the training score remains above 80% if only the top seven features are retained.

From the training scores, using only the seven most important features, it is seen that the top features between the LR, SV and kNN classifiers are similar, with only e_64645 being common to all classifiers.

However, this is based on the single run shown above. Again, running 1000 trials, where *all* top seven features are used for each classifier (`ML-loop_feat.py`), it is seen 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. The LR remains the worst performing classifier for these data, with the SV classifier being best performer, although at the same level (within uncertainties) as the kNN classifier.

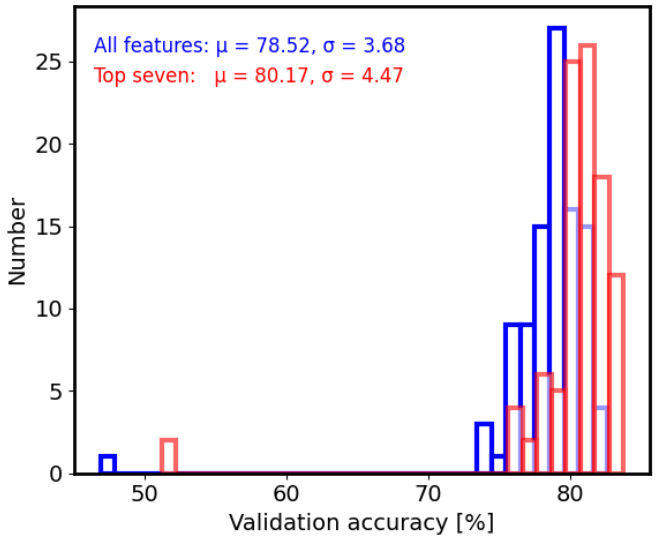
Feature	**Top Seven Classifier Features**				**Number of**
	LR	**SVC**	**kNN**	**DTC**	**Occurrences**
`e_11234`			✓		1
`e_23456`	✓	✓	✓		3
`e_34454`	✓	✓	✓		3
`e_43568`	✓	✓	✓		3
`e_64645`	✓	✓	✓	✓	4
`CountDeposit`				✓	1
`CountPaymentMethod`	✓	✓	✓		3
`CountWithdrawal`				✓	1
`DifferentMethodWithdrawals`	✓			✓	2
`Is_Retail`	✓	✓	✓		3
`IP_Counts`		✓		✓	2
`TotalDeposits`				✓	1
Training 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
Validation accuracy [%]					
LR	74.75 ± 1.63	74.75 ± 1.63	74.31 ± 1.56	73.06 ± 1.64	74.75 ± 1.75
SVC	80.37 ± 1.35	80.27 ± 1.41	80.41 ± 1.35	79.27 ± 1.41	80.85 ± 1.41
kNN	79.99 ± 1.37	79.96 ± 1.39	80.58 ± 1.30	77.99 ± 1.42	80.51 ± 1.34
DTC	78.58 ± 1.65	78.58 ± 1.56	78.46 ± 1.62	74.58 ± 3.26	77.72 ± 1.95

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, I nevertheless test this in case any further insight can be gained.

In `DL.py`, TensorFlow is used with the testing of various hyper-parameters giving the best results for a very simple neural network – just two *ReLu* layers comprising ≈ 50 neurons each.

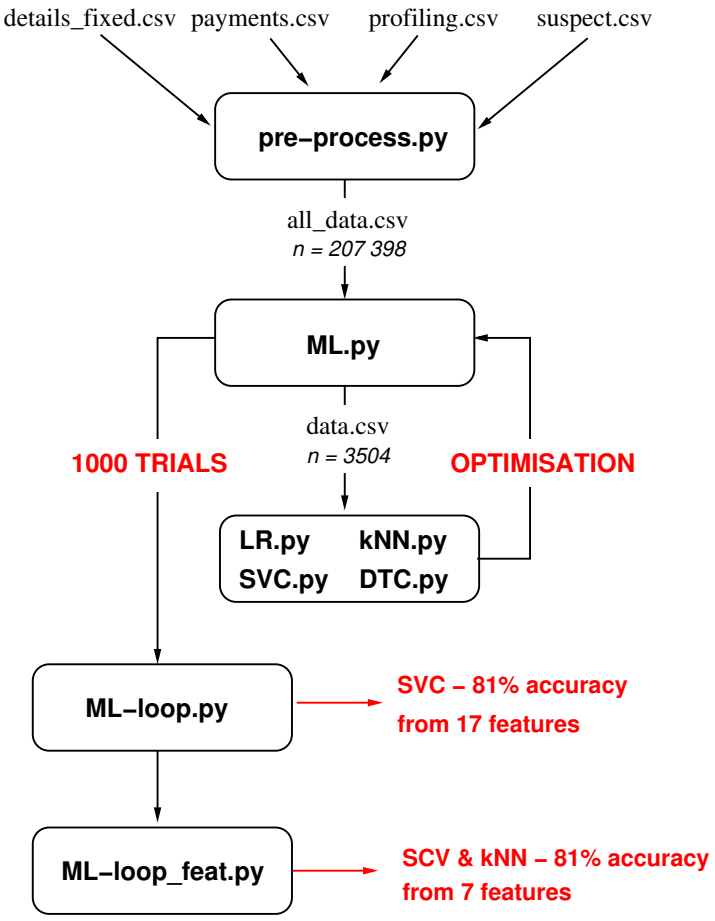
In order to get an accurate representation of the results, in `DL-loop.py` the deep learning algorithm is run 100 times using all of features as well as the top seven features (from the kNN classification, see above). From this it is seen that, although the means of the scores are similar to the machine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



3 Discussion

3.1 Recap

Summarising the steps taken above.



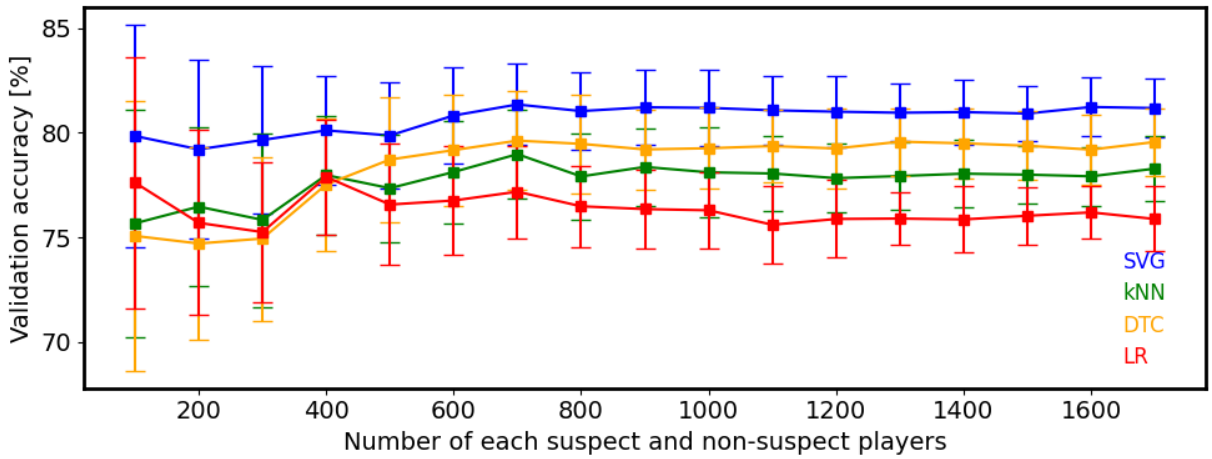
- In `pre-process.py`, the players details (`details_fixed.csv`) are combined with their payment activities (`payments.csv`) and other additional information (`profiling.csv`). Using `suspect.csv`, those with known suspect activity are then flagged.
- In `ML.py`, the 1752 suspect players, in addition to a random sample of 1752 non-suspect, are selected. Four different classification methods 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`, the selection of the 1752 non-suspect players is randomised before re-running the algorithm. After 1000 runs, the Support Vector Classifier is found to be the best performer with an average validation accuracy of $80.9 \pm 1.4\%$.
- Finally, the relative importance of each feature is explored and, testing these, it is apparent that retaining only the seven most important has little detrimental effect. In the case of the SV and kNN classifiers, the performance reaches the same level as the full 17 feature SV 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.[3]

[3]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.

3.2 Prospects

Being trained on just 2×1752 players, the sample is relatively small, as is reflected in the unimpressive deep learning results. Increasing the sample size is expected to increase the accuracy and, in an attempt to quantify this, in `ML_sampled.py` 100 trials for each of the algorithms for randomly selected samples of varying size are tested.



Plotting the results (using `sampled_plot.py`), it appears that the accuracy does not increase significantly for sample sizes above $2\times \approx 500$ players, although the spread in the accuracies narrow and the DT classifier may be trending upwards. In any case, a significant increase in accuracy cannot be ruled out until much larger datasets are used for the training.