AI van de fraude-ontdekking

Een simulatie studie

Jan van Rongen

2022-01-07

0. Inleiding

Door Argos is de in Rotterdam gebruikte "AI" om fraude-gevallen te signaleren gepubliceerd.

Gegeven mijn ervaring met dit soort software leek het me goed de gevolgde methode onder de loep te nemen. Bijzonder daaraan is dat er een voorspellingsmodel wordtr gemaakt met historische data, waarin 60% van de gevallen als fraude zijn beoordeeld. Dat model wordt echter toegepast op de totale populatie van bijstand-trekkers waarvan (volgens andere berichten) maar een zeer beperkt aandeel - slechts enkele procenten - daadwerkelijk fraudeert.

De ervaring leert dat zulke modellen niet goed werken, omdat er veel te weinig gegevens van niet-fraudeurs worden "geleerd". Maar kan dat worden hard gemaakt?

Ter illustratie produceer ik hierbij voorbeelden met behulp van een dataset van ongeveer een kwart miljoen objecten met 1100 kenmerken die diende om koopgedrag te voorspellen. Het dient dus een ander doel, maar de algoritmen zijn hetzelfde,

Dit is afkomstig van een oude Kaggle competitie (Homesite). De training data in die competitie was ongeveer een kwart miljoen records. De prevalentie van de "Ja"-klasse was 18%. Uit deze data isoleren we een nieuwe trainingsset en een "populatie" waarvan we in dit geval dus de klassificatie wel kennen.

1. De aanpak van de simulatie.

1.1 De Data

Elk record in de data bevat een klassificatie (Ja/Nee) en 1110 eigenschappen van het öbject"(in dit geval een persoon) waarover dat record gaat. In het jargon noemt men de klassificatie"target" en de eigenschappen "features". In het Nederlands hebben we het meestal over de doel-variabele en over de kenmerken.

Uit die grote data-set haal ik 13,500 öbjecten met een percentage "Jan" van 60% - de data waarmee het model straks wordt getrained. Vervolgens een data-set van 55,000 waarop we later het model "loslaten". Het percentage "Ja" daarin is 17%. De eerste fase is het fatsoeneren van de input-data en het beoordelen welke kenmerken relevant zijn. De gekozen data was destijds al opgeschoond door ons, dus die stap kunnen we nu overslaan. Welke kenmerken bijdragen aan de voorspellingen heb ik na de eerste goede simulatie vastgesteld met een functie uit de software (`xgb.importance`) waarmee de 200 belangrijkste kenmerken zijn opgevraagd.

1.2. De software van de simulaties.

Net als de Rotterdamse software gebruiken we de statistische programmeertaal R. Ik ga echter niet uitgebreid het "beste" model selecteren, want dat is hier het doel niet. Mijn doel is om te laten zien dat zonder extra maatregelen een model dat is getrained op een 60/40 prevalentie niet goed werkt voor een 17/83 prevalentie in de populatie. Laten we dit fenomeen de **prevalentie-bias** noemen.

In Rotterdam zijn 5 soorten modellen geprobeerd, die elk hun eigen pakket hebben in R: `glmnet`, `gbm`, `xgboost`, `rf` (= Random Forest) en `rpart`.

`glmnet`, `gbm`, `rpart` hebben de mogelijkheid om met de `weights` parameter te proberen de prevalentie-bias weg te werken.

`rf` kent een expliciete parameter `classwt` voor de gewenste prevalentie.

`xgboost` kent twee manieren om de prevalentie-bias te mitigeren. In de eerste plaats kan ook hier een gewicht per observatie worden meegegeven. Dat gebeurt in de matrix structuur van de training data. Daarnaast is er de parameter `scale_pos_weight` die precies doet wat ik wil: de werkelijke verdeling van de Ja-klasse doorgeven.

Ik heb ervoor gekozen om xgboost te gebruiken omdat het het meest flexibele pakket is en ook veel sneller is dan de overige pakketten. En omdat ik dat altijd al gebruikte als eerste analyse-middel.

1.3 Data details

```
##
## Train size: 13500
## Class prevalence: 0.6
```

```
##
## Populatie size: 55000
## Class prevalence: 0.17
```

```
##
## Aantal kenmerken 200
```

1.4 Modellen

We maken vier modellen: één zonder correctie voor de scheve prevalentie en drie met. We voorspellen dan met behulp van het gemaakte model de fraude-precentage van de totale "populatie". Daarvan weten we de klassificatie al en die vergelijken we met de voorspellingen. Alle gevallen mèt correctie werken goed en de enige zonder correctie is slechter dan geen model.

Een noot over de xgboost parameters die we hier gebruiken: dat waren de parameters die in bovenstaande Kaggle competitie goed werkten. Voor de zekerheid hebben we ze gecontroleerd met `caret` – die bevestigt dat.

1.4a. Xgboost zonder correctie

De parameters zijn de volgende:

```
params_a<- list(
  "objective"= "binary:logistic"
  , "booster"= "gbtree"
  , "tree_method" = "hist" ## alles is hier even goed maar deze is de snelste.
  , "eval_metric" = "aucpr" ## net iets beter dan auc
  , "subsample" = 0.90
  , "colsample_bytree" = 0.50
  , "min_child_weight" = 1
  , "max_depth" = 14
  , "gamma" = 9
  , "eta"= 0.02
  , "nthread"= 15

)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 39089  7814
##          1   611  7486
##
##                Accuracy : 0.8468
##                  95% CI : (0.8438, 0.8498)
##     No Information Rate : 0.7218
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.554
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.4893
##             Specificity : 0.9846
##          Pos Pred Value : 0.9245
##          Neg Pred Value : 0.8334
##              Prevalence : 0.2782
##          Detection Rate : 0.1361
##    Detection Prevalence : 0.1472
##       Balanced Accuracy : 0.7369
##
##        'Positive' Class : 1
##
```

1.4b. Xgboost met scale correctie

De parameters zijn de volgende,

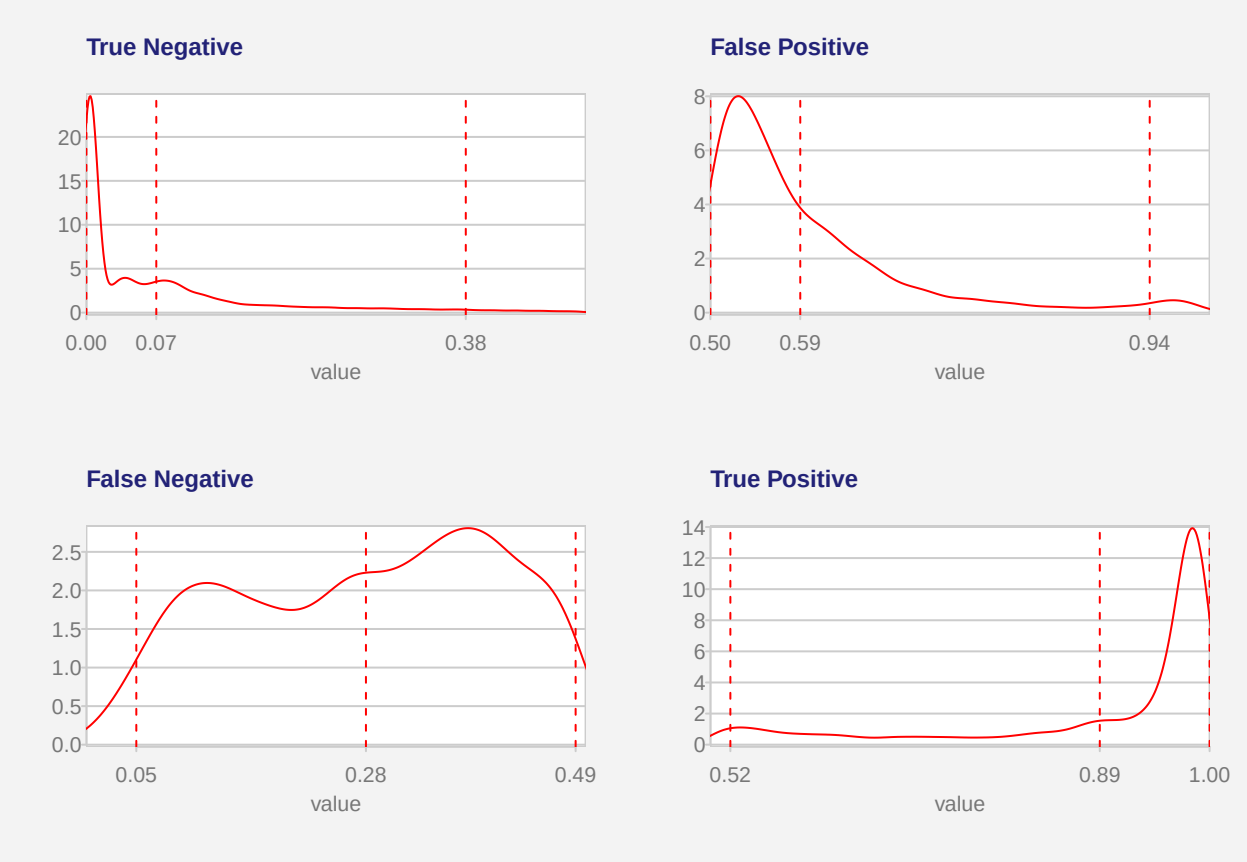
```
params_b<- list(
  "objective"= "binary:logistic"
  , "booster"= "gbtree"
  , "tree_method" = "hist"
  , "eval_metric" = "aucpr" ## slightly better than
                            ## auc or map or ndcg@n)
                            ## or logloss
  , "subsample" = 0.90
  , "colsample_bytree" = 0.5
  , "min_child_weight" = 1
  , "max_depth" = 14
  , "gamma" = 9
  , "eta"= 0.02
  , "nthread"= 15
  , "scale_pos_weight" = 0.15 ##p2 ## = 0.17

)

set.seed(12345)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction     0     1
##          0 46355   548
##          1  3173  4924
##
##                Accuracy : 0.9323
##                  95% CI : (0.9302, 0.9344)
##     No Information Rate : 0.9005
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6888
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.89985
##             Specificity : 0.93594
##          Pos Pred Value : 0.60813
##          Neg Pred Value : 0.98832
##              Prevalence : 0.09949
##          Detection Rate : 0.08953
##    Detection Prevalence : 0.14722
##       Balanced Accuracy : 0.91789
##
##        'Positive' Class : 1
##
```

We analyseren nog even wat verder. Bedenk dat de algoritmes geen absolute scores geven, maar een waarschijnlijkheid. Hoe is dat verdeeld?

Vier diagrammen voor de waarschijnlijkheids-scores bij de Confusion matrix.



De verticale lijnen geven links en rechts het 95% CI aan. De middelste lijn is het gemiddelde.

Dit is geen fraude data maar het gedrag van de "objecten" is wel typisch voor dit soort classificatie-analyses. De True Negatives en tRUE Positives scoren heel dicht bij 0, resp. 1 . De False Positives hebben een net iets te hoge score, maar zijn ver verwijderd van de True positives. En de False Negatives? Dat zijn diegenen met onvoorspelbaar gedrag die dus in geen enkel model wel te vangen zijn.

1.4c. Xgboost met aangepaste gewichten

De parameters zijn de volgende:

```
params_c<- list(
  "objective"= "binary:logistic"
  , "booster"= "gbtree"
  , "tree_method" = "hist"
  , "eval_metric" = "aucpr" ## slightly better than
                            ## auc or map or ndcg@n) or logloss
  , "subsample" = 0.90
  , "colsample_bytree" = 0.50
  , "min_child_weight" = 1
  , "max_depth" = 14
  , "gamma" = 9
  , "eta"= 0.02
  , "nthread"= 15
)

set.seed(12345)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 45729  1174
##          1  2583  5514
##
##                Accuracy : 0.9317
##                  95% CI : (0.9296, 0.9338)
##     No Information Rate : 0.8784
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7068
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8245
##             Specificity : 0.9465
##          Pos Pred Value : 0.6810
##          Neg Pred Value : 0.9750
##              Prevalence : 0.1216
##          Detection Rate : 0.1003
##    Detection Prevalence : 0.1472
##       Balanced Accuracy : 0.8855
##
##        'Positive' Class : 1
##
```

1.4d twee correcties tegelijkertijd

Ik vroeg me af of die twee correcties tegelijk niet tot een over-correctie leidt. Dat valt mee. Juist het aantal fout-positieven daalt sterk en dat zou zeker voordelen kunne hebben.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 46877    26
##          1  4277  3820
##
##                Accuracy : 0.9218
##                  95% CI : (0.9195, 0.924)
##     No Information Rate : 0.9301
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.602
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99324
##             Specificity : 0.91639
##          Pos Pred Value : 0.47178
##          Neg Pred Value : 0.99945
##              Prevalence : 0.06993
##          Detection Rate : 0.06945
```

```
##    Detection Prevalence : 0.14722
##       Balanced Accuracy : 0.95481
##
##        'Positive' Class : 1
##
```

2. Conlusies.

De simulatie in 1,4a laat zien dat het maken van een model zonder rekening te houden met de prevalentie-bias niet (goed) werkt. In dit geval is de accuraatheid zo slecht (0.8468), dat we net zo goed iedereen als niet-fraudeur kunnen bestempelen (0.8528). Immers:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 46903  8097
##          1     0     0
##
##                Accuracy : 0.8528
##                  95% CI : (0.8498, 0.8557)
##     No Information Rate : 0.8528
##     P-Value [Acc > NIR] : 0.503
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.8528
##              Prevalence : 0.1472
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
```

Van de overige schattingen scoort 4b het beste que accuraatheid, maar ook voor 4d valt wat te zeggen als men probeert vooral het aantal fout-positieven te minimaliseren.

3. Management Samenvatting

Deze simulatie toont zeer overtuigend aan dat een model dat gemaakt is met 60/40 prevalentie niet werkt wanneer de werkelijkheid maar hooguit 15% is. Het model dat er uit komt zal een zeer hoog percentage fout-positieven hebben, dat wil zeggen veel mensen aanwijzen als potentiële fraudeur die het niet zijn.

In de praktijk zijn zg. discriminerende kenmerken verwijderd uit het model, maar het is een groot misverstand dat zoiets zou helpen. Dan komen er nl. andere voor in de plaats en _de software blijft zoeken naar een model met 60/40 prevalentie___.

Voorts moet men zich realiseren dat deze algoritmen proberen beide klassen even goed te voorspellen.Een zg. discriminerend kenmerk zou dus heel goed kunnen dienen om juist de niet-fraudeurs te identificeren.

Dat is als de algoritmes worden getrained met representatieve gegevens van beide klassenb, en in de juiste proporties.

Toegift. Wat het had moeten zijn.

Hier laat ik zien wat de juiste aanpak was geweest.

De train_data is 60.40, de populayie misschiem 17/83. Maak een "validation set" uit de train data met die juiste verhoudingen en zet die opzij. Dat heet de hold-out set of validation set. Met de rest gaan we verder als met de vroige train data.

Maar nu vertellen we xgboost dat hij moet trainen met de genoemde data, maar zichzelf moet bijsturen met de validatie-set. Dus de kleinere train set loopt door het algoritme met de bekende verdeling van de validatie set als parameter (die hopelijk die van de populatie is). En dan stopt xgboost op oms verzoek als de doelfunctie zich niet meer verbetert. Werkt dat? Nou uitstekend, want we hebben nu een model gemaakt dateen keurige voorspelling blijkt te doen over de populatie. Laat ik het er op houden: zo had het ook in 2017 gemaakt moeten zijn, dat was toen al lang bekend en mogelijk.

```
## [1]  val_data-auc:0.871028
## Will train until val_data_auc hasn't improved in 750 rounds.
##
## [501]    val_data-auc:0.960334
## [1001]   val_data-auc:0.960762
## [1501]   val_data-auc:0.960850
## [2001]   val_data-auc:0.960888
## [2501]   val_data-auc:0.961049
## [3001]   val_data-auc:0.961118
## [3501]   val_data-auc:0.961140
## [4001]   val_data-auc:0.961184
## Stopping. Best iteration:
## [3577]   val_data-auc:0.961184
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1221   24
##          1   77  178
##
##                Accuracy : 0.9327
##                  95% CI : (0.9188, 0.9448)
##     No Information Rate : 0.8653
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7399
##
##  Mcnemar's Test P-Value : 2.289e-07
##
##             Sensitivity : 0.8812
##             Specificity : 0.9407
##          Pos Pred Value : 0.6980
##          Neg Pred Value : 0.9807
##              Prevalence : 0.1347
##          Detection Rate : 0.1187
##    Detection Prevalence : 0.1700
##       Balanced Accuracy : 0.9109
```

```
##
##        'Positive' Class : 1
##
```

En op de hele populatie:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 45713  1190
##          1  2692  5405
##
##                Accuracy : 0.9294
##                  95% CI : (0.9272, 0.9315)
##     No Information Rate : 0.8801
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6955
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.81956
##             Specificity : 0.94439
##          Pos Pred Value : 0.66753
##          Neg Pred Value : 0.97463
##              Prevalence : 0.11991
##          Detection Rate : 0.09827
##    Detection Prevalence : 0.14722
##       Balanced Accuracy : 0.88197
##
##        'Positive' Class : 1
##
```

====

Jan van Rongen, 2022-01-05