`oxforddown`: An Oxford University Thesis Template for R Markdown

Author Name

Your College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2018

For Yihui Xie

Acknowledgements

This is where you will normally thank your advisor, colleagues, family and friends, as well as funding and institutional support. In our case, we will give our praises to the people who developed the ideas and tools that allow us to push open science a little step forward by writing plain-text, transparent, and reproducible theses in R Markdown.

We must be grateful to John Gruber for inventing the original version of Markdown, to John MacFarlane for creating Pandoc (`http://pandoc.org`) which converts Markdown to a large number of output formats, and to Yihui Xie for creating `knitr` which introduced R Markdown as a way of embedding code in Markdown documents, and `bookdown` which added tools for technical and longer-form writing.

Special thanks to Chester Ismay, who created the `thesisdown` package that helped many a PhD student write their theses in R Markdown. And a very special thanks to John McManigle, whose adaption of Sam Evans' adaptation of Keith Gillow's original maths template for writing an Oxford University DPhil thesis in LaTeX provided the template that I in turn adapted for R Markdown.

Finally, profuse thanks to JJ Allaire, the founder and CEO of RStudio, and Hadley Wickham, the mastermind of the tidyverse without whom we'd all just given up and done data science in Python instead. Thanks for making data science easier, more accessible, and more fun for us all.

Ulrik Lyngs
Linacre College, Oxford
2 December 2018

Abstract

This *R Markdown* template is for writing an Oxford University thesis. The template is built using Yihui Xie's `bookdown` package, with heavy inspiration from Chester Ismay's `thesisdown` and the `OxThesis` LaTeX template (most recently adapted by John McManigle).

This template's sample content include illustrations of how to write a thesis in R Markdown, and largely follows the structure from this R Markdown workshop.

Congratulations for taking a step further into the lands of open, reproducible science by writing your thesis using a tool that allows you to transparently include tables and dynamically generated plots directly from the underlying data. Hip hooray!

Contents

List of Figures

List of Tables

List of Abbreviations

1-D, 2-D . . . One- or two-dimensional, referring **in this thesis** to spatial dimensions in an image.

Otter . . . . . One of the finest of water mammals.

Hedgehog . . . Quite a nice prickly friend.

1 Einleitung

Vergangene und gegenwärtige Bilanzskandale wie ENRON, Worldcom oder Wirecard führen immer wieder zu Diskussionen über die Vertrauenswürdigkeit des Kapitalmarkts sowie über die Vertrauenswürdigkeit der Abschlussprüfung. Bilanzskandale führen weit-reichende Konsequenten mit sich welche eine genauere Analyse des Warums und wie anstoßen (vgl. Boecker/Zwirner 2012, S. 1). Diese Arbeit hat das Ziel, den Einsatz neuronaler Netze gegenüber der gängigeren Me-thode der logistischen Regression (vgl. Bao et al., 2020) zu vergleichen und dabei herauszufinden, ob erstere signifikant besser darin sind, so viele Betrugsfälle wie möglich zu identifizieren, ohne die triviale Annahme zu treffen, dass jeder Fall ein Betrugsfall ist. Bei der Modellevaluation wird angenommen, dass ein nicht-identifizierter Betrugsfall doppelt so schwer wiegt, wie ein prognostizierter Betrugsfall, der sich als Nicht-Betrug herausstellt. Die Metrik zur Messung der Ergebnisse ist der F_β-Score (vgl. Tharwat 2020, S. 174). Dieser bildet das harmonische Mittel aus Präzision und Sensitivität. Die Präzision sagt aus, wie viele vorhergesagte Betrugsfälle tatsächlich Betrugsfälle sind, wobei die Sensi-tivität dabei auf die Frage antwortet, wie viele der tatsächlichen Betrugsfälle als solche identifiziert wurden. Weil die Sensitivität im Kontext der Fraud Detection bedeutungs-voller erscheint, wird sie in dieser Arbeit

doppelt so hoch gewichtet wie die Präzision. Die Formel dieser Metrik lautet dabei:

$$F_\beta = (1 + \beta^2) * \frac{P * R}{\beta^2 * P + R}$$

2

Accounting Fraud und Machine Learning

Das Handelsgesetzbuch (HGB) sieht gemäß § 317 Absatz (Abs.) 1 Satz (S.) 3 vor, dass der Abschlussprüfer im Rahmen der Abschlussprüfung, Unrichtigkeiten und Verstöße welche der Ordnungsmäßigkeit des Abschlusses entgegen stehen, erkennt und entsprechend deklariert. Was unter den Begrifflichkeiten Unrichtigkeit und Verstöße zu verstehen ist, wird durch den Gesetzgeber an dieser Stelle nicht weiter konkretisiert (vgl. Zwernemann et al. 2015, S. 22; § 317 Abs.1 HGB).

Dem entgegen versucht das Institut der Wirtschaftsprüfer (IDW) mit dem veröffentlichten Prüfungsstandard 210 (IDW PS 210) Rechnung zu tragen. Diesem zu entnehmen ist, dass ein Fehlerhafter Abschluss entweder auf Fraud (Verstoß) oder Error (Unrichtigkeit) zurückzuführen ist. Unter dem Begriff Unrichtigkeit wird eine unabsichtliche Angabe im Abschluss verstanden. Konkret bedeutet dies begangene Rechenfehler, eine unbewusst falsche Anwendung von Rechnungslegungsgrundsätzen sowie die falsche Einschätzung von Sachverhalten (vgl. Hlavica et al. 2016, S. 209f.). Der Begriff Verstoß dagegen umfasst eine beabsichtigte Handlung mit dem Ziel rechtswidrige Vorteile zu realisieren. Diese Handlungen konkretisiert der IDW PS 210 als Vermögensschädigungen, Täuschungen und Gesetzesverstöße, welche eine Auswirkung auf die Rechnungslegung zur Folge haben (vgl. Zwernemann et al. 2015, S. 8). Um die Gründe für eine betrügerische Handlung nachvollziehen zu können,

entwickelte Donald Cressey in den 1940er Jahren das sogenannte „Fraud-Triangle". Dieses Dreieck wird ferner dem IDW PS 210 zugrunde gelegt (vgl. Boecker/Zwirner 2012, S. 2f.). Demnach tritt ein Verstoße dann auf, wenn drei Gegebenheiten als erfüllt angesehen werden können. So muss der Täter eine Gelegenheit zu der Tat haben und einen Anreiz (Motivation) für die Tat verspüren. Als letztes muss der Täter die Tat als moralisch akzeptabel rechtfertigen vor sich selbst rechtfertigen (vgl. Schuchter/Levi 2016, S. 3f.). Aber nicht nur durch psychologische Ansätze versucht die Wissenschaft Verstöße einzuordnen und zu identifizieren, sondern auch durch eine Vielzahl an Machine Learning Ansätzen, welche Verstöße mittels Algorithmen identifizieren sollen. Vor dem Hintergrund der potenziellen Gefahren des Bilanzbetrugs werden Letztere zunehmend für die Vorhersage und Aufdeckung von diskretionärer Bilanzpolitik eingesetzt. Die Benchmark in diesem Bereich ist das Dechow et al. Modell, welches auf Grundlage von Accounting and Auditing Enforcement Releases (AAERs) der U.S. Securities and Exchange Comission (SEC) mit Hilfe einer logistischen Regression die Wahrscheinlichkeiten von (bewusst) fehlerhaften Darstellungen schätzt und klassifiziert (Dechow et al. 2011). Hierbei gelten die AAERs als ProxyVariable für die Manipulation der Bilanz. Durch das Voraussetzen der Untersuchungshandlungen seitens der SEC ergibt sich der Vorteil, dass der Typ I Fehler – das Modell sagt fälschlicherweise ein misstatement voraus – deutlich geringer ausfällt. Durch einige wenige Transformationen der logistischen Funktion kann der Einfluss einer jeden unabhängigen Variable durch den entsprechenden Regressionskoeffizienten hinsichtlich der Effektgröße verglichen werden, weswegen die Ergebnisse gut interpretierbar sind. Aus dem Dechow et al. Modell folgt eine korrekte Klassifizierung von misstatements und nonmisstatements von ungefähr 63% (Dechow et al. 2011, S.59). Die Sensitivität, d.h. in wie vielen Fällen das Modell einen misstatement richtig vorhergesagt hat, liegt bei etwas mehr als 68% (339 von 494). Der Typ II Fehler (das Modell klassifiziert ein misstatement als nonmisstatement), der im Rahmen des accounting frauds schwerwiegender ist als der Typ I Fehler (vgl. Lin et al. 2015, S. 468), liegt bei etwas mehr als 31% (155 von 494). Ein Vergleich der Performance der

logistischen Regression mit den neuronalen Netzen, einer weiteren Methode zur Vorhersage von Bilanzbetrug, findet sich in dem Paper von Lin et al. (2015). Aus diesem geht hervor, dass die neuronalen Netze hinsichtlich der Aufdeckung von accounting fraud bessere Ergebnisse liefern als die logistische Regression. Die artificial neural networks (ANNs) erreichen bei dem Testdatensatz eine Genauigkeit von fast 93%. Die Sensitivität von fast 83% liegt zudem deutlich höher als bei der logistischen Regression, bei der 72% der misstatements richtig vorhergesagt wurden (vgl. Lin et al. 2015, S. 465f.). Die Interpretierbarkeit und verhältnismäßig einfache Anwendbarkeit haben die logistische Regression zu einem beliebten Instrument gemacht, die Ergebnisse hinsichtlich der Vorhersage von Bilanzbetrug werden allerdings von anderen Modellen übertroffen (vgl. Dutta et al. 2017, S. 375). Im Falle der neuronalen Netze ergibt sich wiederum der Nachteil der geringeren Transparenz hinsichtlich der Arbeitsweise des Algorithmus (vgl. Bao et al. 2020, S. 228). Schlussendlich ergibt sich ein Trade-Off zwischen der Interpretierbarkeit und Vorhersagekraft. Nachfolgend liegt der Fokus dieser Ausarbeitung auf der reinen Performance respektive Vorhersagekraft des Modells. Ein Modell, welches misstatements richtig vorhersagt, erscheint bei der Klassifizierung von Bilanzbetrug wichtiger als ein Modell, aus dem abgelesen werden kann, welche Variablen den misstatement wie stark beeinflussen. An dieser Stelle sollte erwähnt werden, dass auch ein Modell, welches in 99% der Fälle die richtige Vorhersage trifft, hinsichtlich der Aufdeckung von accounting fraud nicht geeignet sein muss. Durch die Problematik der signifikanten sample imbalance – Betrugsfälle sind stark unterrepräsentiert – ist es möglich, dass misstatements durch das Modell nicht erkannt respektive falsch klassifiziert werden, also Typ II Fehler auftreten können.

3 Datensatz

Ein besonderer Fokus liegt aus diesem Grund auf der Minimierung des Typ II Fehlers respektive der Maximierung der Sensitivität des Modells. Eine höchstmögliche Genauigkeit ist gut, aufgrund der sehr geringen Anzahl an bilanziellen Verfehlungen verglichen zu der Stichprobengröße sollte hier allerdings kein Schwerpunkt liegen. Zum Training und Test des Machine Learning Modells, wird ein Datensatz aus der Veröffentlichung von Bao et al. (2020) verwendet. Die Autoren haben diesen im Internet auf der Seite „GitHub“ zur Verfügung gestellt (vgl. Bao et al. 2020, GitHub Repository). Dieser besteht aus allen öffentlich gelisteten US-amerikanischen Firmen im Zeitraum von 1991 bis 2008. Die Accounting-Betrugsfälle aus den „Accounting and Auditing Enforcement Releases“ (AAER), die von der United States Securities and Exchange Commission (SEC) im gleichen Zeitraum veröffentlicht worden sind (vgl. Bao et al. 2020, S. 207). Der Datensatz listet für jeden Eintrag 28 verschiedene finanzielle Items auf. Diese setzen sich aus den Veröffentlichungen von Cecchini et al. (2010) und Dechow et al. (2011) zusammen. Die finanziellen Items stammen aus vier verschiedenen Bereichen, der Bilanz (z.B. gesamte Forderungen), der Gewinn- und Verlustrechnung (z.B. Nettoumsatz), der Kapitalflussrechnung (z.B. Langzeitemission von Schuldtiteln) und dem Marktwert (z.B. Common Shares Outstanding). Da Accounting-Betrugsfälle eher seltener vorkommen (vgl. Dutta et

al. 2011, S. 381), weist der Datensatz eine große Verteilungsungleichheit zwischen den Betrugs- und Nicht-betrugsfällen auf. Weniger als ein Prozent aller Einträge im Datensatz sind hierbei Betrugsfälle. Weitere Betrugsfälle könnten über andere Datenbanken gesucht werden, hierbei gibt es allerdings das Problem, dass die Betrugsfälle identifiziert und einzeln herausgesucht werden müssen. Daher besteht hier ein „class imbalance“ Problem, das mithilfe des Machine Learning Algorithmus gelöst werden muss. Zudem sind nur Betrugsfälle bis zum Jahr 2008 in diesem Datensatz erhalten, da die SEC nach der Finanzkrise ihre Prioritäten geändert hat (vgl. Bao et al. 2020, S. 208). Sollte sich die Art und Weise mit der Accounting-Betrug durchgeführt wird in den Jahren danach geändert haben, so können diese Fälle unter Umständen nicht vom Algorithmus erkannt werden.

AB HIER HANDELT SICH DER AUFSCHRIEB MEHR UM NOTIZEN ALS UM EINE ABGABEFÄHIGE VERSION.

Cleaning-Prozess-Reichenfolge:

1. Daten als “data” Laden
2. Spalten “p_aaer” und “new_p_aaer” löschen
3. Alle Zeilen mit NaN-Werten löschen
4. all_data bilden: Besteht nur aus 14 + 28 + 2 Vars
5. all_data via Jahreszahl normalisieren. fyear droppen.
6. raw_data (28 + 1 Vars) und ratio_data (14 + 1 Vars) aus all_data bilden
7. deskriptive Statistiken können mit den Datensätzen: min, Max, Mean, Median 0.25 und 0.75 Quantile UND normalisierte Boxplots und Histogramme. Ggf noch Ausreißer raus.

Dann ist die Datenvorbereitung fertig und man kann Modelle damit rechnen. Bis zu dem Punkt geht die nächste Zelle:

```
## 1.
data <- import("data/uscecchini28.csv")
```

```r
## 2.
data <- data[,-match(c("p_aaer", "new_p_aaer"), names(data))]

## 3.
data <- data[complete.cases(data),]

## 4.
all_names <- c("fyear", "misstate", "act", "ap", "at", "ceq", "che",
               "cogs", "csho", "dlc", "dltis", "dltt", "dp", "ib",
               "invt", "ivao", "ivst", "lct", "lt", "ni", "ppegt",
               "pstk", "re", "rect", "sale", "sstk", "txp", "txt",
               "xint", "prcc_f", "dch_wc", "ch_rsst", "dch_rec",
               "dch_inv", "soft_assets", "dpi", "ch_cs", "ch_cm",
               "ch_roa", "ch_fcf", "reoa", "EBIT", "issue", "bm")
all_data <- data[, match(all_names, names(data))]

## 5.
# Funktion schreiben
normalize <- function(x){
  return((x - min(x)) / (max(x) - min(x)))
}
# Funktion anwenden
for (year in unique(all_data$fyear)){
  for (col in names(all_data)){
    all_data[data$fyear == year, col] <- normalize(
      all_data[data$fyear == year, col]
      )
  }
}
# drop fyear-Variable
```

```
all_data <- all_data[, -1]

## 6.
raw_names <- c("misstate", "act", "ap", "at", "ceq", "che", "cogs",
               "csho", "dlc", "dltis", "dltt", "dp", "ib", "invt",
               "ivao", "ivst", "lct", "lt", "ni", "ppegt", "pstk",
               "re", "rect", "sale", "sstk", "txp", "txt", "xint",
               "prcc_f")

ratio_names <- c("misstate", "dch_wc", "ch_rsst", "dch_rec",
                 "dch_inv", "soft_assets", "dpi", "ch_cs", "ch_cm",
                 "ch_roa", "ch_fcf", "reoa", "EBIT", "issue", "bm")

raw_data <- all_data[, match(raw_names, names(all_data))]
ratio_data <- all_data[, match(ratio_names, names(all_data))]

# ================================================================| Maximale Br

## 7. Statistiken erstellen. SKIP.
```

4 Methoden

Diese Arbeit vergleicht die Vorhersagequalität der in der Accounting Fraud Detection gängigen logistischen Regression nach Dechow et al. (vgl. Bao et al. 2020, S. 2) mit der von neuronalen Netzen. Die logistische Regression ist eng mit der linearen Regression verwandt und wird zur binären Schätzung einer Klassenzugehörigkeit verwandt (vgl. Géron 2019, S. 144). Dabei berechnet sie eine gewichtete Summe von Inputfaktoren und aggregiert sie zu einer Wahrscheinlichkeit zwischen 0 und 1. Liegt diese Wahrscheinlichkeit bei mindestens 0.5, so wird die Klasse „1“ vorhergesagt, welcher in dieser Arbeit der Klasse „fraud“ entspricht (vgl. Géron 2019, S. 144). Hierzu wird für jedes Attribut einer Beobachtung eine Sigmoid-Funktion verwendet, welche S-förmig vom Minimum bis zum Maximum des jeweiligen Attributs verläuft und die Verteilungen der Merkmalsausprägungen möglichst gut nach Klassenzugehörigkeit abgrenzt. Je weiter eine Merkmalsausprägung von dieser Grenze entfernt ist, desto näher ist der Funktionswert an der 1 oder der 0 (vgl. Géron 2019, S. 148). Neuronale Netze bestehen aus drei Sorten von Schichten: Input-Schichten, welche Daten einlesen, versteckte Schichten, welche die Daten verarbeiten und Output-Schichten, welche aus den verarbeiteten Daten eine Prognose ableiten (vgl. Géron 2019, S. 286). In dieser Arbeit besteht die Output-Schicht aus lediglich einem Knoten, welche die Klassen „fraud“ abbildet. Die Anzahl der Knoten der

Input-Schicht entspricht der Anzahl der Variablen im Datensatz. Die Knoten einer Schicht sind jeweils mit jedem Knoten seiner nachfolgenden Schicht durch „Gewichte“ verbunden. Jeder einzelne Knoten aggregiert die Signale, die er empfängt über deren Gewichte zu einer Zahl und wendet eine Aktivierungsfunktion an (vgl. Géron 2019, S. 282). Übersteigt der Funktionswert einen gegeben Schwellenwert, so „feuert“ das Neuron, was bedeutet, dass es ein Signal größer 0 an die Neuronen der nächsten Schicht weitergibt (vgl. Géron 2019, S. 282-283). Die Gewichte und alle Schwellenwerte werden durch Backpropagation unter Zuhilfenahme des Gradient Descent Algorithmus verbessert (vgl. Géron 2019, S.119 und S, 286). Sind alle Trainingsdaten einmal zum Training herangezogen worden, bedeutet das, dass das Netz für „eine Epoche“ trainiert wurde (vgl. Géron 2019, S. 127). In dieser Arbeit wird ein Netz über 100 Epochen hinweg trainiert.

AB HIER HANDELT SICH DER AUFSCHRIEB MEHR UM NOTIZEN ALS UM EINE ABGABEFÄHIGE VERSION.

TODO:

1. Evaluate-Funktion bilden
2. LogReg mit allen Daten. Ablauf:
    1. Train-test-Split (X_train, y_train, X_test, y_test)
    2. SOMTE in X_train anwenden, seed = 42
    3. Training
    4. Prediction (in y_pred speichern)
    5. Evaluate-Funktion anwenden
3. LogReg mit Rohdaten
    1. ...
4. LogReg mit Ratios
    1. ...
5. Summary von allen 3 Modellen

```
## 1. EVALUATE BILDEN
evaluate <- function(test, pred, border = 0.5){

  pred <- ifelse(pred > border, 1, 0)
  confusion <- table(test, pred)
  TN <- confusion[1,1]
  TP <- confusion[2,2]
  FP <- confusion[1,2]
  FN <- confusion[2,1]

  total_acc <- numeric(2)
  total_acc[1] <- NaN
  total_acc[2] <- round((TN + TP) / sum(confusion),4)

  prec <- numeric(2)
  prec[1] <- NaN
  prec[2] <- round(TP / (TP + FP),4)

  sens <- numeric(2)
  sens[1] <- NaN
  sens[2] <- round(TP / (TP + FN),4)

  F1 <- numeric(2)
  F1[1] <- NaN
  F1[2] <- round(2*(prec[2]*sens[2])/(prec[2] + sens[2]), 4)

  F.score <- function(beta, p = prec[2], s = sens[2]){
    round((1 + beta^2)*(p*s)/(beta^2*p + s),4)
  }
```

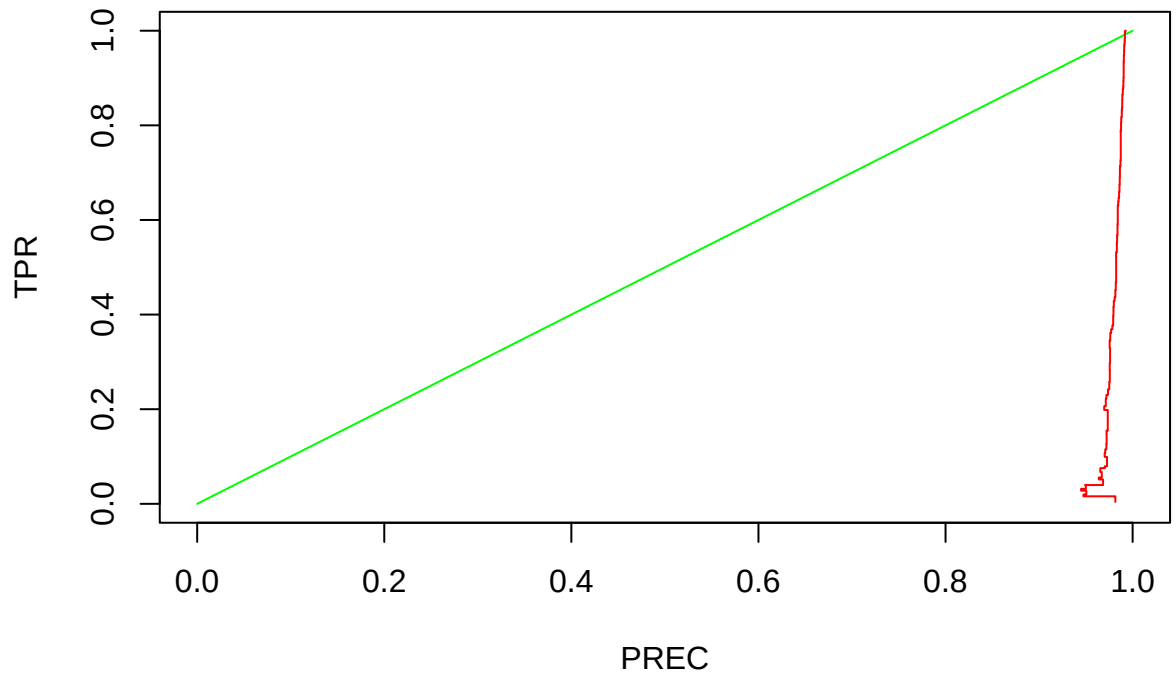
```
  F2 <- numeric(2)
  F2[1] <- NaN
  F2[2] <- F.score(2)

  F.5 <- numeric(2)
  F.5[1] <- NaN
  F.5[2] <- F.score(0.5)

  return(cbind(confusion, total_acc, prec, sens, F1, F2, F.5))

}
```

```
## 2. MODELL MIT ALLEN DATEN TRANIEREN
train_test_split_smote<- function(data, y_col, frac = 0.7, seed = 42,
                                  k = 5){
  set.seed(seed)

  smp_size <- floor(frac * nrow(data))
  train_ind <- sample(seq_len(nrow(data)), size = smp_size)

  X_smote <- data[train_ind, -match(y_col, names(data))]
  y_smote <- data[train_ind, match(y_col, names(data))]

  true_frac <- sum(y_smote) / length(y_smote)

  train_smote_object <- SMOTE(X_smote, y_smote, K = 5,
                              dup_size = 1 / true_frac)$data

  X_train <- train_smote_object[,-match('class',
                                        names(train_smote_object))]
```

```
  y_train <- as.numeric(train_smote_object[,
                        match('class', names(train_smote_object))])

  X_test <- data[-train_ind, -match(y_col, names(data))]
  y_test <- data[-train_ind, match(y_col, names(data))]

  return(list(X_train = X_train, X_test = X_test, y_train = y_train,
              y_test = y_test))
}

splitted_data <- train_test_split_smote(data = all_data,
                                        y_col = 'misstate', frac = 0.7)

X_train <- splitted_data$X_train
X_test <- splitted_data$X_test
y_train <- splitted_data$y_train
y_test <- splitted_data$y_test

train <- X_train
train$misstate <- y_train

logit_all_data <- glm(misstate ~., data = train,
                      family = "binomial")
y_pred_logit_all_data <- predict.glm(logit_all_data,
                                     X_test, type = "response" )

evaluate(y_test, y_pred_logit_all_data)
```

```
##       0     1 total_acc   prec   sens     F1     F2    F.5
## 0 24068 10624       NaN    NaN    NaN    NaN    NaN    NaN
## 1    91   161    0.6934 0.0149 0.6389 0.0291 0.0681 0.0185
```

```
tpr <- c()
pre <- c()
for(i in seq(0.001, 0.999, 0.01)){
  res <- evaluate(y_test, y_pred_logit_all_data, border = i)
  pre <- c(pre, 1 - res[2, "prec"])
  tpr <- c(tpr, res[2, "sens"])
}

TPR <- seq(0,1,0.1)
PREC <- seq(0,1,0.1)

plot(PREC, TPR, type="l", col="green")
lines(pre, tpr, type = "s", col = "red")
```



```
summary(logit_all_data)
```

```
##
## Call:
## glm(formula = misstate ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.5406  -0.9641   0.1000   0.9745   5.5802
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -23.75983    0.50318 -47.219  < 2e-16 ***
## act          11.12570    1.41279   7.875 3.41e-15 ***
## ap            1.98894    0.49944   3.982 6.82e-05 ***
## at           10.80001    0.82025  13.167  < 2e-16 ***
## ceq          -5.65817    0.16604 -34.078  < 2e-16 ***
## che          -3.62274    0.73212  -4.948 7.49e-07 ***
## cogs         29.99856    1.55593  19.280  < 2e-16 ***
## csho          6.41974    0.27687  23.187  < 2e-16 ***
## dlc         -12.09188    0.76046 -15.901  < 2e-16 ***
## dltis        -2.55895    0.39298  -6.512 7.43e-11 ***
## dltt         17.01835    0.72114  23.599  < 2e-16 ***
## dp           -6.59251    0.65110 -10.125  < 2e-16 ***
## ib           -0.24714    0.13140  -1.881  0.06000 .
## invt         -3.59020    0.53125  -6.758 1.40e-11 ***
## ivao          1.37159    0.43429   3.158  0.00159 **
## ivst         -2.32630    0.48566  -4.790 1.67e-06 ***
## lct          18.48910    1.38337  13.365  < 2e-16 ***
## lt          -35.46112    1.37644 -25.763  < 2e-16 ***
## ni            1.39985    0.12153  11.518  < 2e-16 ***
```

```
## ppegt        -6.34845    0.65147  -9.745  < 2e-16 ***
## pstk         -7.67877    0.35223 -21.800  < 2e-16 ***
## re            4.27251    0.05994  71.281  < 2e-16 ***
## rect          2.51693    0.80673   3.120  0.00181 **
## sale        -35.36934    1.76975 -19.986  < 2e-16 ***
## sstk          1.57673    0.23121   6.819 9.15e-12 ***
## txp          -3.70640    0.36741 -10.088  < 2e-16 ***
## txt          -0.55736    0.06326  -8.811  < 2e-16 ***
## xint          6.73976    0.38732  17.401  < 2e-16 ***
## prcc_f       12.19458    0.26323  46.327  < 2e-16 ***
## dch_wc       -1.97133    0.12683 -15.543  < 2e-16 ***
## ch_rsst       3.04583    0.15251  19.971  < 2e-16 ***
## dch_rec       1.61872    0.06807  23.779  < 2e-16 ***
## dch_inv       1.30387    0.06491  20.089  < 2e-16 ***
## soft_assets   2.03327    0.02586  78.635  < 2e-16 ***
## dpi          -0.49160    0.06295  -7.809 5.76e-15 ***
## ch_cs         0.98542    0.09663  10.198  < 2e-16 ***
## ch_cm        -0.57042    0.10192  -5.597 2.18e-08 ***
## ch_roa       -2.12709    0.12845 -16.559  < 2e-16 ***
## ch_fcf        1.91612    0.19656   9.748  < 2e-16 ***
## reoa         22.60563    0.59876  37.754  < 2e-16 ***
## EBIT         -3.30312    0.20996 -15.732  < 2e-16 ***
## issue         0.66310    0.02341  28.324  < 2e-16 ***
## bm           -0.39105    0.08582  -4.557 5.19e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 225660  on 162783  degrees of freedom
```

```
## Residual deviance: 184472  on 162741  degrees of freedom
## AIC: 184558
##
## Number of Fisher Scoring iterations: 7
```

```
## 2. MODELL MIT ROH-DATEN TRANIEREN
splitted_data <- train_test_split_smote(data = raw_data,
                                        y_col = 'misstate', frac = 0.7)
X_train <- splitted_data$X_train
X_test <- splitted_data$X_test
y_train <- splitted_data$y_train
y_test <- splitted_data$y_test

train <- X_train
train$misstate <- y_train

logit_raw_data <- glm(misstate ~., data = train,
                      family = "binomial")
y_pred_logit_raw_data <- predict.glm(logit_raw_data, X_test,
                                     type = "response" )

evaluate(y_test, y_pred_logit_raw_data)
```

```
##       0     1 total_acc   prec   sens     F1     F2    F.5
## 0 24616 10076       NaN    NaN    NaN    NaN    NaN    NaN
## 1   115   137    0.7084 0.0134 0.5437 0.0262 0.061 0.0166
```

```
summary(logit_raw_data)
```

```
##
## Call:
## glm(formula = misstate ~ ., family = "binomial", data = train)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.7581  -1.0411   0.0837   1.1359   3.3212
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.04527    0.01501 -69.648  < 2e-16 ***
## act           8.96766    1.37982   6.499 8.08e-11 ***
## ap            5.63238    0.50424  11.170  < 2e-16 ***
## at           20.18742    0.94107  21.451  < 2e-16 ***
## ceq          -4.94838    0.15049 -32.882  < 2e-16 ***
## che          -7.32059    0.73408  -9.972  < 2e-16 ***
## cogs         25.58273    1.52700  16.754  < 2e-16 ***
## csho          5.23000    0.26738  19.560  < 2e-16 ***
## dlc          -9.52638    0.74396 -12.805  < 2e-16 ***
## dltis        -2.06434    0.36976  -5.583 2.37e-08 ***
## dltt         16.47397    0.72063  22.861  < 2e-16 ***
## dp           -8.53646    0.65377 -13.057  < 2e-16 ***
## ib           -0.05232    0.11352  -0.461 0.644863
## invt         -1.16582    0.51706  -2.255 0.024151 *
## ivao          0.96618    0.44619   2.165 0.030358 *
## ivst         -1.72751    0.52092  -3.316 0.000912 ***
## lct          14.80146    1.36620  10.834  < 2e-16 ***
## lt          -38.23713    1.37870 -27.734  < 2e-16 ***
## ni            1.27856    0.10622  12.037  < 2e-16 ***
## ppegt       -13.45517    0.71929 -18.706  < 2e-16 ***
## pstk         -9.80106    0.36253 -27.035  < 2e-16 ***
## re            3.64993    0.05327  68.517  < 2e-16 ***
## rect          5.92899    0.79359   7.471 7.95e-14 ***
## sale        -31.50988    1.74106 -18.098  < 2e-16 ***
```

```
## sstk          1.79127    0.23203   7.720 1.16e-14 ***
## txp          -4.02413    0.36272 -11.094  < 2e-16 ***
## txt          -1.08172    0.05855 -18.475  < 2e-16 ***
## xint          5.42623    0.37607  14.429  < 2e-16 ***
## prcc_f       11.17267    0.23612  47.317  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 225660  on 162783  degrees of freedom
## Residual deviance: 205190  on 162755  degrees of freedom
## AIC: 205248
##
## Number of Fisher Scoring iterations: 6
```

```
## 3. MODELL MIT RATIO-DATEN TRANIEREN
splitted_data <- train_test_split_smote(data = ratio_data,
                                        y_col = 'misstate', frac = 0.7)

X_train <- splitted_data$X_train
X_test <- splitted_data$X_test
y_train <- splitted_data$y_train
y_test <- splitted_data$y_test

train <- X_train
train$misstate <- y_train

logit_ratio_data <- glm(misstate ~., data = train,
                        family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
y_pred_logit_ratio_data <- predict.glm(logit_ratio_data, X_test,
                                       type = "response" )


evaluate(y_test, y_pred_logit_ratio_data)
```

```
##       0     1 total_acc   prec   sens    F1     F2    F.5
## 0 20526 14166       NaN    NaN    NaN   NaN    NaN    NaN
## 1    91   161     0.592 0.0112 0.6389 0.022 0.0523 0.0139
```

```
summary(logit_ratio_data)
```

```
##
## Call:
## glm(formula = misstate ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -3.156  -1.083   0.527   1.013   6.492
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -35.74613    0.56539 -63.224  < 2e-16 ***
## dch_wc       -2.84428    0.12496 -22.762  < 2e-16 ***
## ch_rsst       7.46171    0.14249  52.366  < 2e-16 ***
## dch_rec       1.77243    0.06538  27.110  < 2e-16 ***
## dch_inv       1.16130    0.06228  18.645  < 2e-16 ***
## soft_assets   1.99318    0.02350  84.816  < 2e-16 ***
## dpi          -0.25691    0.05951  -4.317 1.58e-05 ***
## ch_cs         1.00253    0.09622  10.419  < 2e-16 ***
## ch_cm        -1.03301    0.09577 -10.787  < 2e-16 ***
## ch_roa       -4.42682    0.13154 -33.653  < 2e-16 ***
## ch_fcf        9.45417    0.18045  52.392  < 2e-16 ***
```

```
## reoa         33.46111    0.67111  49.860  < 2e-16 ***
## EBIT         -4.38108    0.22272 -19.670  < 2e-16 ***
## issue         0.93498    0.02250  41.562  < 2e-16 ***
## bm           -1.20690    0.07980 -15.124  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 225660  on 162783  degrees of freedom
## Residual deviance: 200018  on 162769  degrees of freedom
## AIC: 200048
##
## Number of Fisher Scoring iterations: 7
```

Appendices

A

The First Appendix

This first appendix includes an R chunk that was hidden in the document (using `echo = FALSE`) to help with readibility:

In 02-rmd-basics-code.Rmd

And here's another one from the same chapter, i.e. Chapter ??:

B

The Second Appendix, for Fun