```
knitr::opts_chunk$set(echo = TRUE)
```

The Sarbanes-Oxley Act: Security, Privacy and Fraud Threats to Firm Systems

The dataset that was used in the analysis in Sarbanes-Oxley (SOX) data reports breaches for 213 of the 3398 listed firms which report SOX assessments between 2005 and 2015. Restriction of the analysis to subsets in figure 2 public firms that experienced a breach resolves the issue with firms by dropping the breached firms for which we have no SOX 404 reports. This yields a conceptual model for the panel regression of the form where represent security breach classifications and other metrics from the Privacy Clearinghouse / Verizon dataset and represents SOX 302 and 404 reported assessments from the Audit Analytics dataset. Datasets were dichotomized into manual and

The rapid initial decline in the SOX adverse percentage rate for auditor attestations from an initial 16.9 to a current level of around 2 might suggest that SOX 404 audit procedures which are estimated to cost around $2 million annually are not adding significant information about systems security over that provided by management in internal auditors at the firm in their self-reported SOX 302 attestations. To investigate we tested : SOX 404 attestations add new information to managements self-reported SOX 302 attestations. This tests the veracity of some of the critiques discussed in the introduction that SOX 404 attestations are costly without providing information not already available in SOX 302 attestations.

```
library(kableExtra)

tbl <- read.csv("/home/westland/Documents/SOX-AI/sox_dataset_contents.csv")
colnames(tbl)=c("Item", "Number of Items")

knitr::kable(tbl, "latex", booktabs = T, caption = "Contents of the
Dataset") %>%
  kable_styling(full_width=T) %>%
  column_spec(1, width = "15cm") %>%
  column_spec(2, width = "5cm")
```

I tested a panel of binomial models logit and probit (figure 3) using a General Linear Model but achieved better fit using standard linear model panel regression. The reason that a binomial model may be preferred is to avoid estimator bias with smaller sample sizes but the current chapter's data sets are sufficiently large that there is no advantage to using probit or logit models and linear model panel regression approaches will yield unbiased estimators. Test statistics for all the models indicate there are possible unobserved individual effects and thus we used fixed effects panel regression. An example of logit model fit, using the base-R `glm` function appears in the following code chunk.

```
library(readr)
library(tidyverse)
```

```
sox_data <-
  read_csv("/home/westland/Documents/SOX-AI/innerjoinSOXBRCH.csv",
           col_types = cols(X1 = col_skip())) %>%
  select(
    IC_IS_EFFECTIVE,
    MATERIAL_WEAKNESS,
    SIG_DEFICIENCY,
    AUDITOR_AGREES,
    CARD.x
    )

sox_card_logit <- glm(CARD.x~., family=binomial(link=logit), data=sox_data)
summary(sox_card_logit)
```

Notice in the regression results that that the signs of coefficients for 'internal control effectiveness' and 'material weakness' and 'significant deficiency' are opposite, because the first is a statement about the effectiveness of controls, while the latter are about ineffectiveness of controls. AIC values are small, suggesting good fit from using the logit model.

We concluded that SOX 404 auditors maintain their independence consistent with Generally Accepted Auditing Standard (GAAS). With the investigation of potentially confounding factors of size and auditor independence effectively completed we can move on to testing our main hypothesis: that SOX 404 and 302 attestations contain information about the future occurrence of specific types of real world security breaches.

Using an Autoencoder to Detect Control Weaknesses

Autoencoders are self-supervised machine learning models, where the generated targets are the input, unmodified. Since only a small percentage of the transactions in the SOX-breach data set represent credit card fraud at the audited corporations, we have a highly unbalanced classification problem. Traditional classification approaches are a poor choice for decision making because the dataset contains such a small sample of frauds. Similarly, linear regression models suffer from singular design matrices, and may fail to capture non-linear relationships in real-world data and predictors are multicolinear, which creates singularities.

To address these problems, I have designed a bottleneck autoencoder model for dimensionality reduction to explore the SOX feature space, similar to the use of principal component analysis. Because of the multicolinearity of accounting datasets, machine learning methods that train many weights in multiple layers are likely to extract this information more effectively to generate a single component which should provide a surrogate for 'control weakness'. An autoencoder is a neural network that is used to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. For an autoencoder to work well we have a strong initial assumption: that the distribution of variables for normal firms is different from the distribution for ones with control weaknesses. Since security breaches should have a different distribution then normal transactions (because their reporting is heavily manipulated by criminals and victimized firms) we expect that our autoencoder will have higher reconstruction errors on security breaches then on normal transactions. This means that we can use the reconstruction error as a quantity that indicates whether a clients systems are more likely to suffer security breaches. I use the

audencoder's dimensionality reduction to explore the dataset's feature space (similar to what to we could do with a principal component analysis, as in figure 5).
I designed two autoencoders, one in `Tensorflow / Keras` for reference, and one in `H2O` for analysis of SOX control weaknesses, `H2O` (https://www.h2o.ai/) is a popular open source machine learning platform accessible in R which runs both in-memory and on `H2O` servers, offering an interface that is slightly simpler to use than Tensorflow's. `H2O` supports the most widely used statistical & machine learning algorithms including gradient boosted machines, generalized linear models, deep learning and so forth, and contains an *AutoML* function that automatically runs through all the algorithms and their hyperparameters to produce a leaderboard of the best models. For larger models, `H2O4GPU` provides GPU-accelerated machine learning on hardware platforms with, e.g., NVidia's GPUs.

As is typical with accounting data, predictors are multicolinear, which creates problems for traditional regression approaches because of the singularities. Figure 5's cursory plot of principal components (a linear clustering algorithm) reveals that most of the variance is weighted on the first principal component. This does not mean that the data doesn't convey information; it's just that the information involves more complex multicovariate non-linear relationships. Because of this, machine learning methods that train many weights in multiple layers are likely to extract this information more effectively. This single component is likely to be a surrogate for 'control weakness', suggesting covariance between SOX reports and occurrence of security breaches (otherwise, we would likely see clustering around two large principle components).

```
scree_sox <- read.csv("/home/westland/Documents/SOX-
AI/innerjoinSOXBRCH.csv")
scree_sox <- scree_sox[,-c(1:2)]
scree_sox[is.na(scree_sox)] <- 0
screeplot(prcomp(scree_sox),
          main="The \"Control Weakness\" Principle Component ",
          xlab="Component"
          )
```

Figure 5 shows the multicolinearity of the combined SOX and Privacy Clearinghouse information -- most of the information in both datasets can be captured in one principle component. Given that SOX reports control weaknesses, and security breaches occur because of control weaknesses, this principle component is a strong surrogate for formal definitions of "control weakness".

Preprocessing

Preprocessing splits the dataset into train and test sets and then *min-max* normalize the data (this is done because neural networks work much better with small input values). Based on the `date` variable, we will use the results before 2011 for training and the rest for testing. This is good practice because when using the model we want to predict future breaches based on SOX reports issued previously.

Two helper functions are used to normalize the database (i.e., adjust the values to similar scale, which helps the machine learning algorithms converge more quickly). The are not really needed for this data, which is binary $[0,1]$ data, but if you wish to reuse this code on other data, they will be useful. The helper functions include:

1. a function to get descriptive statistics about the dataset that are used for min-max scaling (using the same normalization constants for training and test sets), and
2. a function to perform the min-max scaling.

```
library(tidyverse)
library(purrr)
library(keras)

## function to get descriptive statistics

get_desc <- function(x) {
  map(x, ~list(
    min = min(.x),
    max = max(.x),
    mean = mean(.x),
    sd = sd(.x)
  ))
}

## function to normalize values

normalization_minmax <-
  function(x, desc) {
  map2_dfc(
    x,
    desc,
    ~(.x - .y$min)/(.y$max - .y$min))
}


## get data
sox <-
  read_csv(
    "~/Documents/SOX-AI/innerjoinSOXBRCH.csv",
    col_types = cols(X1 = col_skip())
    ) %>%
  select(
    date,
    CARD.x,
    IS_EFFECTIVE,
    MATERIAL_WEAKNESS,
    SIG_DEFICIENCY,
    IC_OP_TYPE,
    AUDITOR_AGREES,
    COMBINED_IC_OP,
    IC_IS_EFFECTIVE,
    AUDIT_FEES
  )
sox[is.na(sox)] <- 0

## create normalized datasets for ML
```

```
desc <- sox %>% get_desc()

x_train <-
  sox %>%
  filter(date <= 2011) %>%
  normalization_minmax(desc)  %>%
  select(-c(date))

x_test <-
  sox %>%
  filter(date > 2011) %>%
  normalization_minmax(desc)  %>%
  select(-c(date))

y_train <-
  x_train[,"CARD.x"] %>%   ## CARD.x the fraud indicator
  as.matrix()

y_test <-
  x_test[,"CARD.x"] %>%
  as.matrix()

x_train <-
  x_train %>%
  select(-c(CARD.x)) %>%
  as.matrix()

x_test <-
  x_test %>%
  select(-c(CARD.x)) %>%
  as.matrix()
```

Tensorflow Implementation of the Autoencoder

Here is the reference autoencoder in `Tensorflow / Keras`. The model converges quickly and accurately, suggesting that SOX control weakness reporting can be a successful predictor of vulnerability to credit card fraud at client firms.

```
## assure that the Keras API is available

library(keras)
install_keras()
```

```
## define the model in Keras,
## this is a symmetric autoencoder with 4 fully connected layers.
```

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 100, activation = "tanh", input_shape =
ncol(x_train)) %>%
  layer_dense(units = 10, activation = "tanh") %>%
  layer_dense(units = 100, activation = "tanh") %>%
  layer_dense(units = ncol(x_train))

summary(model)


## the model is a binary classifier, so we use an entropy loss

model %>% compile(
  loss = "binary_crossentropy",
  optimizer = "adam",
  metrics= 'accuracy'
)

checkpoint <- callback_model_checkpoint(
  filepath = "model.hdf5",
  save_best_only = TRUE,
  period = 1,
  verbose = 1
)

early_stopping <- callback_early_stopping(patience = 5)

history <- model %>% fit(
  x = x_train[y_train == 0,],
  y = x_train[y_train == 0,],
  epochs = 10,
  batch_size = 32,
  validation_data = list(x_test[y_test == 0,], x_test[y_test == 0,]),
  callbacks = list(checkpoint  , early_stopping)
)

plot(history)
```

The H2O implementation of autoencoders and anomaly detection for fraud analytics

The *h2o* package (https://www.h2o.ai/) provides the R interface to the H2O open source machine learning platform which supports gradient boosted machines, generalized linear models, deep learning and models. It has become popular due to *AutoML* which runs through algorithms and their hyperparameters to produce a leaderboard of the best models. It works on a wide range of big data infrastructures, Hadoop or Spark clusters and can directly read files from HDFS, Spark, S3, Azure Data Lake and a variety of other data sources into it’s in-memory distributed key value store, and is relatively easy to productize. R also offers a

GPU-accelerated H2O package *h2o4gpu* package. The following example applies H2O to industry wide security assessment using SOX data.

The SOX-card fraud dataset needs special treatment when performing machine learning because they are severely unbalanced, Only ~$3.44 % $ of audits were for companies that experienced a credit card fraud, affirming that the database is highly unbalanced.

```
cat("% of SOX reports where client had credit fraud during the year = ",
100*sum(sox$CARD.x)/length(sox$CARD.x), "% of audits")
```

In such cases, the optimal predictor would predict in $100 %$ of audits that there would be no credit card frauds, and that predictor would be right a respectable $96.56 %$ of the time. A closer look at audit fee distributions and dates of frauds suggests that fraud occurrence is not random, and suggests there is information to be extracted from the dataset.

```
library(tidyverse)

sox %>%
  ggplot(aes(x = AUDIT_FEES)) +
    geom_bar(color = "grey", fill = "lightgrey") +
    theme_bw() +
    scale_y_continuous(trans='log2') +
    facet_wrap( ~ CARD.x, scales = "free", ncol = 2)


sox %>%
  ggplot(aes(x = date)) +
    geom_bar(color = "grey", fill = "lightgrey") +
    theme_bw() +
    facet_wrap( ~ CARD.x, scales = "free", ncol = 2)
```

Interestingly, fraudulent credit card transactions had much more variability in the audit fees charged. This could have many causes, but it also seems to suggest that where fraud is a possibility, there may be greater uncertainty in the client's controls that necessitates additional audit steps.

For modeling, I used R's H2O implementation and converted the dataset to H2O format, splitting the dataset into training and test sets.

```
library(h2o)
h2o.init(nthreads = -1)

creditcard_hf <- as.h2o(sox)
```

```
splits <- h2o.splitFrame(creditcard_hf,
                         ratios = c(0.30, 0.30),
                         seed = 123)

train_unsupervised  <- splits[[1]]
train_supervised  <- splits[[2]]
test <- splits[[3]]

response <- "CARD.x"
features <- setdiff(colnames(train_unsupervised), response)
```

In `H2O` I start by training an unsupervised autoencoder neural network by setting `autoencoder = TRUE`. Similar to the `Tensorflow` model, I use a bottleneck model that reduces the dimensionality of the data down to 2 nodes/dimensions. The autoencoder then + learns which credit card transactions are similar and which transactions are outliers or anomalies. Autoencoders can be sensitive to outliers, and thus may be prone to overfitting.

```
library(h2o)

model_nn <- h2o.deeplearning(x = features,
                             training_frame = train_unsupervised,
                             model_id = "model_nn",
                             autoencoder = TRUE,
                             reproducible = TRUE,
                             seed = 123,
                             hidden = c(10, 2, 10),
                             epochs = 100,
                             activation = "Tanh")

model_nn
#Convert to autoencoded representation
test_autoenc <- h2o.predict(model_nn, test)
```

Bottleneck autoencoders offer the ability to extract features captured in the middle layers, similar to factor analysis or PCA, but the features need not be linear nor must they adhere to the simple optimization of traditional exploratory statistics. . We can extract hidden features with the h2o.deepfeatures() function and plot to show the reduced representation of the input data.

```
library(tidyverse)

train_features <- h2o.deepfeatures(
  model_nn,
  train_unsupervised,
  layer = 2
  ) %>%
  as.data.frame() %>%
  mutate(card_fraud = as.vector(train_unsupervised[, "CARD.x"]))

ggplot(
  train_features,
  aes(x = DF.L2.C1, y = DF.L2.C2, color = card_fraud)) +
  geom_point(alpha = 0.1)
```

Graphing the two features extracted from layer 2 of the autoencoder shows that feature `DF.L2.C1` clearly dichotomizes audits into those with high potential for card fraud, and those without. Dimensionality reduction through an autoencoder model alone can clearly identify fraud in this dataset. This concept can be carried forward further, using the reduced dimensionality representation of one of the hidden layers as features for model training. An example would be to use the 10 features from the first or third hidden layer:

```
#  Take the third hidden layer

train_features <-
  h2o.deepfeatures(
    model_nn,
    train_unsupervised,
    layer = 3
    ) %>%
  as.data.frame() %>%
  mutate(card_fraud = as.factor(as.vector(train_unsupervised[, "CARD.x"])))
%>%
  as.h2o()

response="card_fraud"

features_dim <- setdiff(colnames(train_features), response)

model_nn_dim <- h2o.deeplearning(y = response,
                               x = features_dim,
                               training_frame =  train_features,
                               reproducible = TRUE,
                               balance_classes = TRUE,
                               seed = 123,
                               hidden = c(10, 2, 10),
                               epochs = 100,
                               activation = "Tanh")


h2o.saveModel(model_nn_dim, path="model_nn_dim", force = TRUE)

model_nn_dim
```

One way to explore the performance of this model is to plot the 'gains' chart. Gain at a given percentile is the ratio of cumulative number of targets (e.g., frauds) up to that percentile to the total number of targets (e.g., frauds) in the entire data set. Here is a gains chart of the model.

```
library(tidyverse)
library(knitr)
library(kableExtra)
library(ggplot2)

## Gains/Lift Tables

gains <- h2o.gainsLift(model_nn_dim,train_features, valid=F, xval=F)


gains %>%
  ggplot(aes(y=gains$gain, x=gains$cumulative_data_fraction)) +
  geom_line() +
  xlab("percentile") + ylab("gain") +
  xlim(0,1)
```

```
# For measuring model performance on test data, we need to
# convert the test data to the same reduced dimensions as the # trainings
data:


test_dim <- h2o.deepfeatures(model_nn, test, layer = 3)

h2o.predict(model_nn_dim, test_dim) %>%
  as.data.frame() %>%
  mutate(actual = as.vector(test[, "CARD.x"])) %>%
  group_by(actual, predict) %>%
  summarise(n = n()) %>%
  mutate(freq = n / sum(n))
```

This looks quite good in terms of identifying fraud cases: 91% of fraud cases were identified! However, many non-fraud cases were also classified as fraud. In actual audit applications this would not be a good model, and it is necessary to look further to improve the sensitivity and specificity of predictions..

Anomaly detection

We can also ask which instances were considered outliers or anomalies within our test data, using the `h2o.anomaly()` function. Based on the autoencoder model that was trained before, the input data will be reconstructed and for each instance, the mean squared error (MSE) between actual value and reconstruction is calculated for both class labels.

```
anomaly <- h2o.anomaly(model_nn, test) %>%
  as.data.frame() %>%
  tibble::rownames_to_column() %>%
  mutate(
    card_fraud =
      as.factor(
        as.vector(
          test[, "CARD.x"]
          )))


mean_mse <- anomaly %>%
  group_by(card_fraud) %>%
  summarise(mean = mean(Reconstruction.MSE))


## This, we can now plot:

ggplot(anomaly, aes(x = as.numeric(rowname), y = Reconstruction.MSE, color
= as.factor(card_fraud))) +
  geom_point(alpha = 0.3) +
  geom_hline(data = mean_mse, aes(yintercept = mean, color = card_fraud)) +
  scale_color_brewer(palette = "Set1") +
  labs(x = "instance number",
       color = "card_fraud")
```

The plot does not show a clear-cut classification into fraud and non-fraud cases but the mean MSE is definitely higher for fraudulent transactions than for regular ones.

We can now identify outlier instances by applying an MSE threshold for what we consider outliers. We could e.g. say that we consider every instance with an MSE > 0.02 (chosen according to the plot above) to be an anomaly/outlier.

```
anomaly <- anomaly %>%
  mutate(outlier = ifelse(Reconstruction.MSE > 0.02, "outlier",
"no_outlier"))

anomaly %>%
  group_by(card_fraud, outlier) %>%
  summarise(n = n()) %>%
  mutate(freq = n / sum(n))
```

For this dataset, outlier detection is insufficient to correctly classify fraudulent credit card transactions either. This suggests that further analysis is necessary. In this example, we will continue with supervised learning to attempt to use SOX data to predict credit card fraud.

Pre-trained supervised model

In this section, the autoencoder model is used as a pre-training input for a supervised model. This model will use the weights from the autoencoder for model fitting.

```
train_features <- h2o.deepfeatures(
  model_nn,
  train_unsupervised,
  layer = 2
  ) %>%
  as.data.frame() %>%
  mutate(card_fraud = as.vector(train_unsupervised[, "CARD.x"]))
```

```
response <- "CARD.x"

features <- setdiff(colnames(train_supervised), response)

model_nn_2 <- h2o.deeplearning(y = response,
                               x = features,
                               training_frame = train_supervised,
                               pretrained_autoencoder  = "model_nn",
                               reproducible = TRUE,
                               balance_classes = F,
                               ignore_const_cols = TRUE,
                               seed = 123,
                               hidden = c(10, 2, 10),
                               epochs = 100,
                               activation = "Tanh")

model_nn_2
```

```
pred <- as.data.frame(h2o.predict(object = model_nn_2, newdata = test)) %>%
  mutate(actual = as.vector(test[, "CARD.x"]))

pred %>%
  group_by(actual, predict) %>%
  summarise(n = n()) %>%
  mutate(freq = n / sum(n))

summary(pred)
```

This shows promise as the means are close (though the model tends to skew towards underprediction of fraud). This suggests that it is worthwhile to further improve the model by e.g. performing grid search for hyperparameter tuning, going back to the original features, selecting different engineered features and perhaps exploring different algorithms.

Measuring model performance on highly unbalanced data

Because of the severe bias towards non-fraud cases, we can not use performance measures like accuracy or area under the curve (AUC), as they would give overly optimistic results based on the high percentage of correct classifications of the majority class. An alternative to AUC is to use the precision-recall curve or the sensitivity (recall)-specificity curve. To calculate and plot these metrics, we can use the ROCR package. There are different ways to calculate the area under a curve (e.g., see the PRROC package for details). In the next analysis I create a simple line-integral function that calculates the area between every consecutive points-pair of $x$ (i.e.$ x_1 - x_0, x_2 - x_1$, etc.) under the corresponding values of $y$.

```
library(ROCR)

line_integral <- function(x, y) {
  dx <- diff(x)
  end <- length(y)
  my <- (y[1:(end - 1)] + y[2:end]) / 2
  sum(dx * my)
}

str(pred)
summary(pred)

prediction_obj <- prediction(pred$predict, pred$actual)


par(mfrow = c(1, 2))
par(mar = c(5.1,4.1,4.1,2.1))

# precision-recall curve
perf1 <- performance(prediction_obj, measure = "prec", x.measure = "rec")

x <- perf1@x.values[[1]]
y <- perf1@y.values[[1]]
```

```
y[1] <- 0

plot(perf1, main = paste("Area Under the\nPrecision-Recall Curve:\n",
round(abs(line_integral(x,y)), digits = 3)))

# sensitivity-specificity curve
perf2 <- performance(prediction_obj, measure = "sens", x.measure = "spec")

x <- perf2@x.values[[1]]
y <- perf2@y.values[[1]]
y[1] <- 0

plot(perf2, main = paste("Area Under the\nSensitivity-Specificity
Curve:\n", round(abs(line_integral(x,y)), digits = 3)))
```

Precision is the proportion of test cases predicted to be fraud that were indeed fraudulent (i.e. the true positive predictions), while recall or sensitivity is the proportion of fraud cases that were identified as fraud. And specificity is the proportion of non-fraud cases that are identified as non-fraud.

The precision-recall curve tells us the relationship between correct fraud predictions and the proportion of fraud cases that were detected (e.g. if all or most fraud cases were identified, we also have many non-fraud cases predicted as fraud and vice versa). The sensitivity-specificity curve thus tell us the relationship between correctly identified classes of both labels (e.g. if we have 100% correctly classified fraud cases, we will have no correctly classified non-fraud cases and vice versa).

We can also look at this a little bit differently, by manually going through different prediction thresholds and calculating how many cases were correctly classified in the two classes:

```
thresholds <- seq(from = 0, to = 1, by = 0.1)
pred_thresholds <- data.frame(actual = pred$actual)

for (threshold in thresholds) {

  prediction <- ifelse(pred$predict > threshold, 1, 0)
  prediction_true <- ifelse(pred_thresholds$actual == prediction, TRUE,
FALSE)
  pred_thresholds <- cbind(pred_thresholds, prediction_true)

}

colnames(pred_thresholds)[-1] <- thresholds
pred_thresholds %>%
  gather(x, y, 2:ncol(pred_thresholds)) %>%
  group_by(actual, x, y) %>%
  summarise(n = n()) %>%
  ggplot(aes(x = as.numeric(x), y = n, color = actual)) +
    geom_vline(xintercept = 0.6, alpha = 0.5) +
    geom_line() +
    geom_point(alpha = 0.5) +
    theme_bw() +
    facet_wrap(actual ~ y, scales = "free", ncol = 2) +
    labs(x = "prediction threshold",
         y = "number of instances")
```

Figure 13's plots tell us that we can increase the number of correctly classified non-fraud cases without loosing correctly classified fraud cases when we increase the prediction threshold from the default 0.5 to 0.6:

```
pred %>%
  mutate(predict = ifelse(pred$predict > 0.6, 1, 0)) %>%
  group_by(actual, predict) %>%
  summarise(n = n()) %>%
  mutate(freq = n / sum(n))
```

Our final model does not greatly improve on the previous models, suggesting at this point that more information is needed for accurate prediction. Auditors in particular should have access to more extensive proprietary information than the publicly reported data in SOX. This can conceivably be merged with the SOX data, while using the previous analysis to provide additional specificity and sensitivity in credit card fraud detection.