Investigating Frankel and Rose (1996)

Goals

1. Replicate Frankel and Rose (1996) which is a simple probit model but choose probability cutoff to maximise the area under the ROC curve.
2. Show off-the-shelf machine learning on the same dataset: random forest, KNN and SVN

Summary of findings

1. Probit models often do not need tuning but in the context of imbalanced dataset, we can make a probit model much more sensitive by setting the threshold of classifying a crisis observation low. For instance, as opposed to using a 50% cut-off (i.e. when predicted probability exceeds 0.5, classify the obs as crisis), you can use other cut-off values. Below I used the value to maximise the area under the ROC curve.
2. After making this change, the gap between probit and machine learning algos shrinks although random forest and KNN still do better. Mind you without "tuning" the probit model, its performance is nowhere close to its ML counterparts.

Note to self

1. The main difficulty of this prediction task comes from the imbalance of classes - we have more non-crisis obs than crisis.
2. Of course one way to do it is to oversample the minority class… which I did in this script, but it just feels unnatural to just generate more data points.
3. In the literature, quite a few papers are horse races of conventional machine learning and almost all manage to show ML algos do better. But of course they are - they are built for predictions.
4. So to stand out, you either come up with a more interpretable model or find a novel way to deal with the class imbalance, or indeed come up with even more advanced models.
5. Intepretability of ML models is big among the ai guys. While the goal here is not casuality but still we will want to know which variable is more important in shaping the forecast. And there are quite a few libraries out there in python that do this: scikit learn plus tons others.
6. When it comes to a novel way to deal with class imbalance, a natural fit (and it so happens it's one of the models fewer people know of) is an autoeoncoder which can be paired with LSTM and CNN. The idea is simple, just train the model using the non-crisis data, then you will have a good idea of how things play out in no crisis. Then reconstruct the whole dataset using the trained model. Then what you should see is that reconstruction error is low where there is no crsis, and shoots up when there is one. This way you do not have to oversample the dataset to solve the class imbalance.
7. Another additional benefit is that with LSTM you also make use of the time dimension in classification.

Replicate Frankel and Rose (1996)

Data

I managed to find the underlying data from Rose's personal website (http://faculty.haas.berkeley.edu/arose/RecRes.htm, scroll down to "Banking and Exchange Crises in Developing Countries"). He also provided the STATA file used for data-cleaning. A summary of what they did is provided in the next section.

```
#replicating Frankel and Rose (1996)
frankel_rose_data <- read.dta("./stata/cleanrose12.dta")
```

Data cleaning

Frankel and Rose (1996) estimated a probit model in an attempt to explain the incidence of currency crisis for over 100 developing countries across between 1971 to 1992. In their paper, a currency crisis is defined by 2 criteria:

1. Nominal exchange rate increases by 25% or more in a year; and,
2. The rate of increase is at least 10% or more than the growth rate in the previous year.

In order to avoid double-counting a crisis, crises that are within 3 years of each other are excluded from the sample. This results in 117 episodes of currency crises which are plotted in figure 1.

The list of explanatory variables used is shown as follows:

1. **comrat**: Commercial bank debt (as % of total debt)
2. **conrat**: Concessional debt (as % of total debt)
3. **varrat**: Variable debt (as % of total debt)
4. **fdistock**: FDI stock
5. **shorttot**: Short-term debt (as % of total debt)
6. **pubrat**: Public sector debt (as % of total debt)
7. **multirat**: Multilateral debt (as % of total debt)
8. **debty**: Total debt (as % of GNP)
9. **reservem**: Ratio of international reserves to monthly imports
10. **cacc**: Current account (as % of GDP)
11. **defrat**: Government position (as % of GNP)
12. **dlcred**: Percentage growth of domestic credit
13. **dly**: Percentage growth of per capita GNP
14. **istar**: Foreign interest rate
15. **overaln**: Real exchange rate divergence (over-valuation)

The STATA script that does all these transformations can be found in `stata/rose_dat_cleaning.do`

Figure 1: 117 episodes of currency crisis identified in Frankel and Rose (1996)

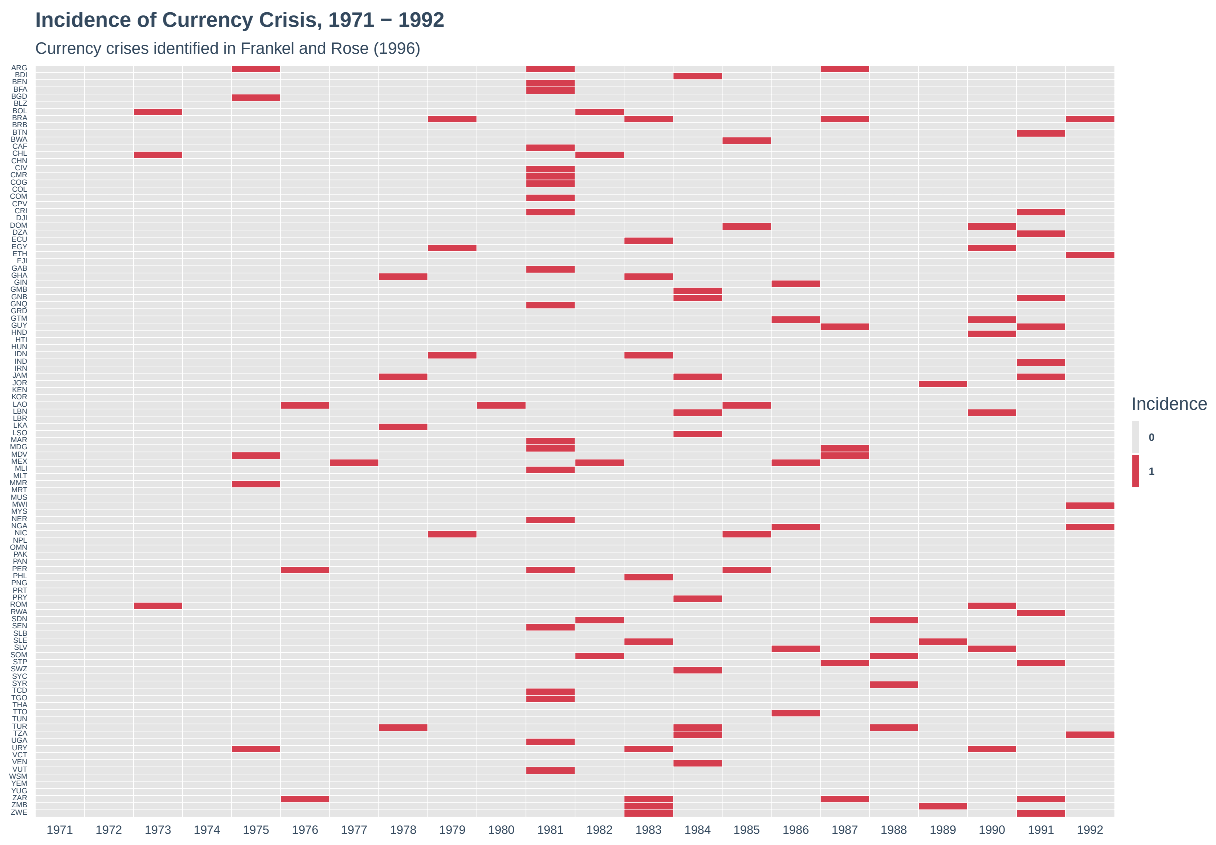
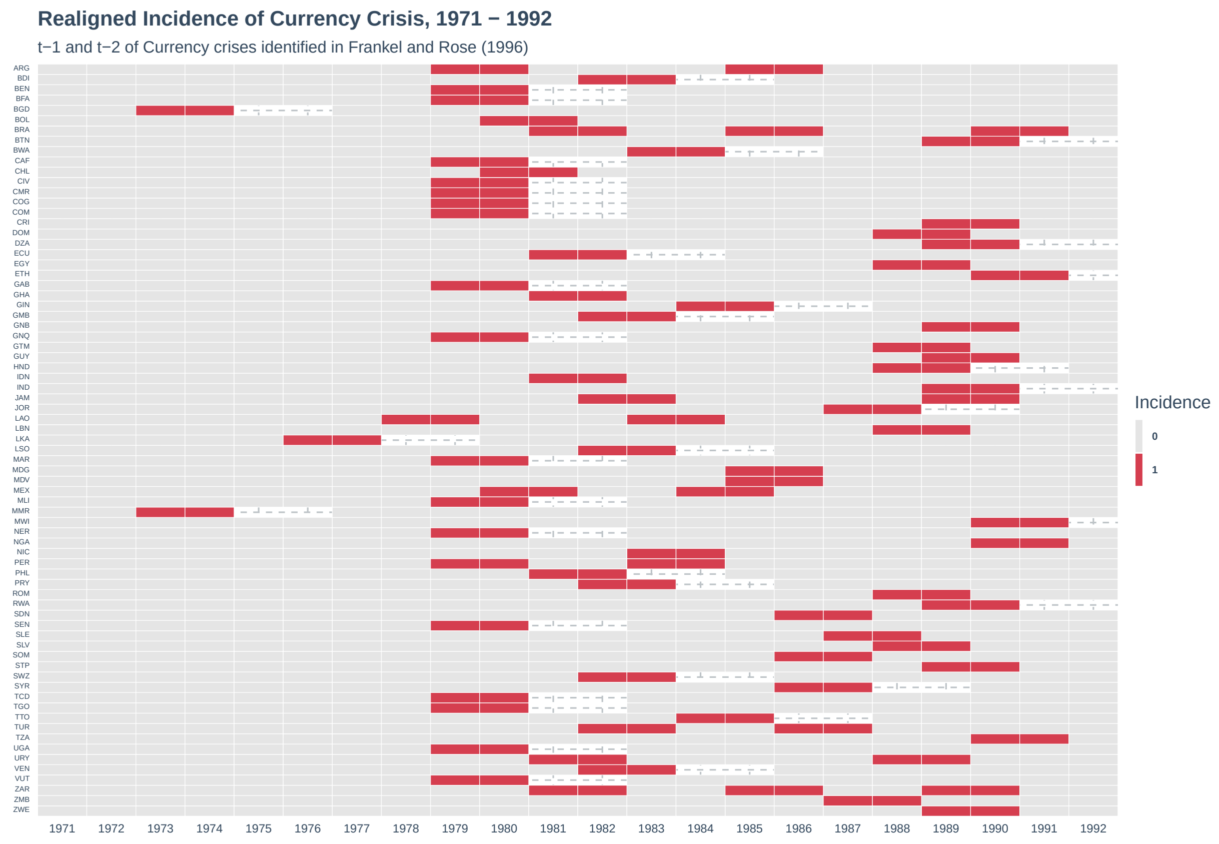
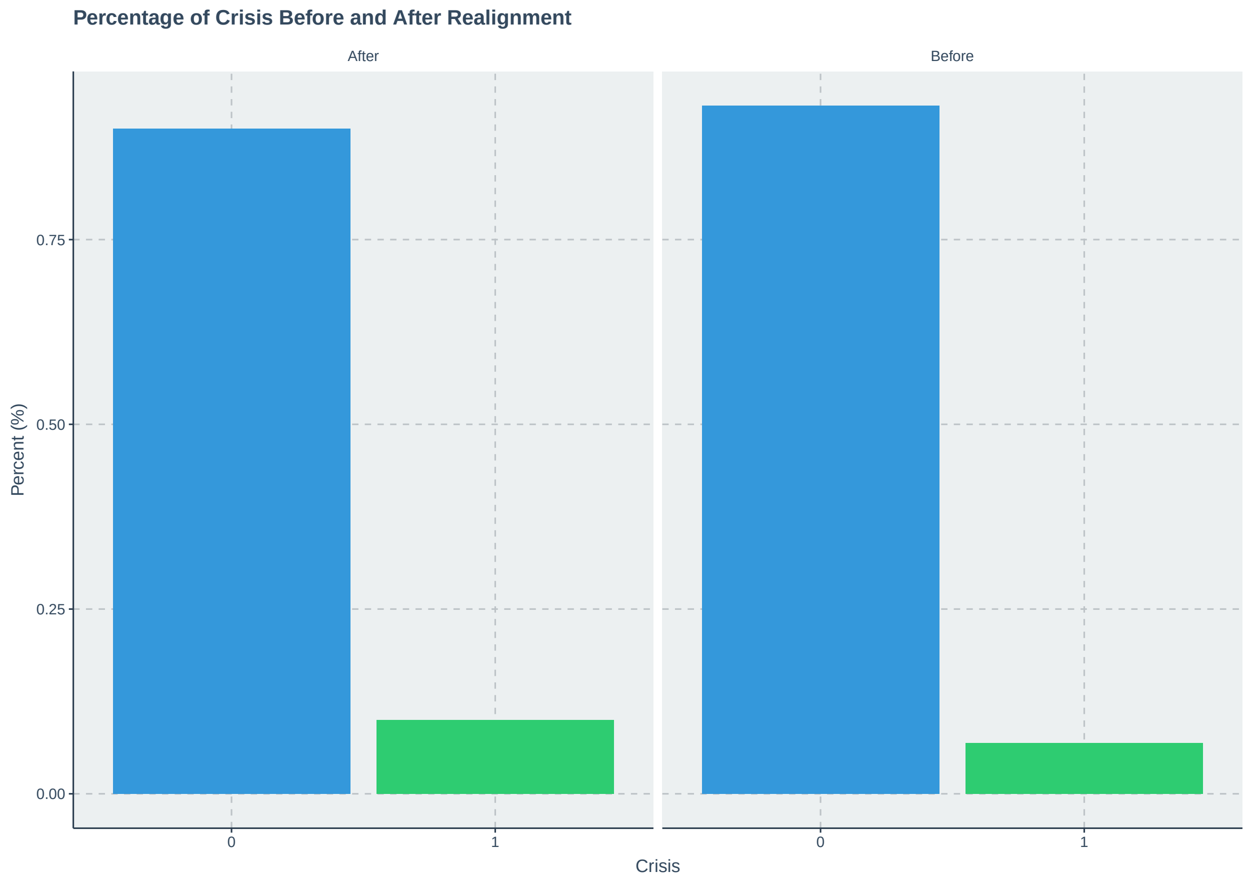


Figure 2: Realigned Incidence of Currency Crisis, 1971 - 1992



After the re-alignment, the percentage of crisis observations goes up from 6% to around 11%.

Figure 3: Percentage of Crisis Before and After Realignment



Replication

A close, *but not exact*, replication of tge probit results in Frankel and Rose (1996) is shown in table 1.

```
# run a simple probit model here
frankel_rose_probit <- glm(event ~ comrat + conrat + varrat + fdistock + shorttot + pubrat +
                               multirat + debty + reservem + cacc + defrat + dlcred + dly +
                               istar + overvaln, family = binomial(link = "probit"),
                data = frankel_rose_data)
```

Table 1: Baseline probit estimates in Frankel and Rose (1996)

	Dependent variable:
	event
comrat	-0.005
	(0.013)
conrat	-0.010^{*}
	(0.006)
varrat	0.001
	(0.013)
fdistock	-0.031^{**}
	(0.013)
shorttot	0.004
	(0.012)
pubrat	0.011
	(0.008)
multirat	-0.003
	(0.007)
debty	0.003
	(0.002)
reservem	-0.001^{**}
	(0.0003)
cacc	0.011
	(0.010)
defrat	0.027^{**}
	(0.014)
dlcred	0.013^{***}
	(0.003)
dly	-0.037^{***}
	(0.012)
istar	0.115^{***}
	(0.025)
overvaln	0.005
	(0.003)
Constant	-2.542^{***}
	(0.898)
Observations	803
Log Likelihood	-191.346
Akaike Inf. Crit.	414.693
Note:	*p<0.1; **p<0.05; ***p<0.01

Moving into machine learning

Create a score card

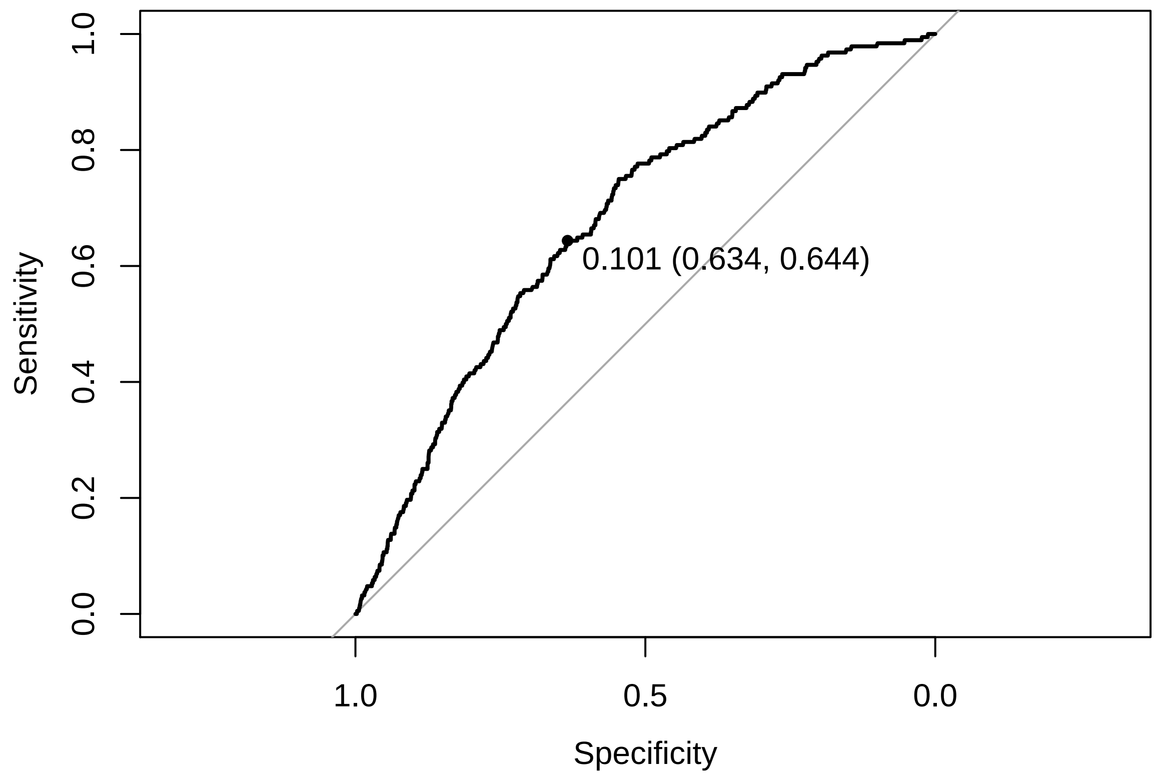
```r
records = matrix(NA, nrow = 4, ncol=2)
colnames(records) <- c("train.error", "test.error")
rownames(records) <- c("Probit", "KNN", "Random Forests", "SVM")
```

Partition the data

```r
frankel_rose_data_ML <- frankel_rose_data_realigned %>%
  mutate(event = ifelse(is.na(event), 0, event),
         event = factor(event, levels = c("0", "1"), labels = c("no", "yes"))) %>%
  # keep a smaller subset, simplify subsequent codes
  select(event, comrat, conrat, varrat,
         fdistock, shorttot, pubrat, multirat, debty,
         reservem, cacc, defrat, dlcred, dly, istar, overvaln)

#frankel_rose_data_ML_cleaned <- frankel_rose_data_ML[complete.cases(frankel_rose_data_ML), ]

# generate row indices for training data
set.seed(2019)
in_train <- createDataPartition(frankel_rose_data_ML$event,
                                p = 0.8, list=FALSE)

training <- frankel_rose_data_ML[in_train, ]
test <- frankel_rose_data_ML[-in_train, ]

# imputing missing variables using bagged trees
# an easy tutorial can be found here:
# http://rismyhammer.com/ml/Pre-Processing.html

# covariates live in columns 2-16
preProc <- preProcess(method = "bagImpute", training[, 2:16])
training[, 2:16] <- predict(preProc, training[, 2:16])
test[, 2:16] <- predict(preProc, test[, 2:16])
```

Probit

```r
probit_fit <- glm(event ~ ., family = binomial(link = "probit"), data = training)

# Training
prob_train <- predict(probit_fit, type = "response")

# Quick look at the ROC
# Do it by hand for now, can write a function later

prob_train_roc <- roc(training$event, prob_train)

# this shows the roc curve
plot(prob_train_roc, print.thres = "best", print.thres.best.method = "closest.topleft")
```



```
# look for the best top-left cutoff
prob_train_coords <- coords(prob_train_roc, "best", best.method = "closest.topleft", ret = c("threshold

# Choose the best cut-off
prob_train_labels = training %>%
  mutate(predicted.value = factor(ifelse(prob_train <= prob_train_coords$threshold, "no", "yes"), level

probit_train_error <- calc_error_rate(predicted.value = prob_train_labels$predicted.value, true.value =

# Test
prob_test <- predict(probit_fit, newdata = test, type = "response")

prob_test_labels = test %>%
  mutate(predicted.value = factor(ifelse(prob_test <= prob_train_coords$threshold, "no", "yes"), levels

probit_test_error <- calc_error_rate(predicted.value = prob_test_labels$predicted.value, true.value = t

# write down the training and test errors of the probit model
records[1,] <- c(probit_train_error, probit_test_error)
```

Entering Machine Learning

Oversampling technique

Here I show the code that oversamples the minority class (i.e. having a crisis) using the algorithm from Chawla et al. (2002)

Before:

```
table(training$event)
```

```
##
##   no  yes
## 1689  188
```

After oversampling:

```
set.seed(2019)
smote_train <- SMOTE(event ~ ., data = training)
table(smote_train$event)
```

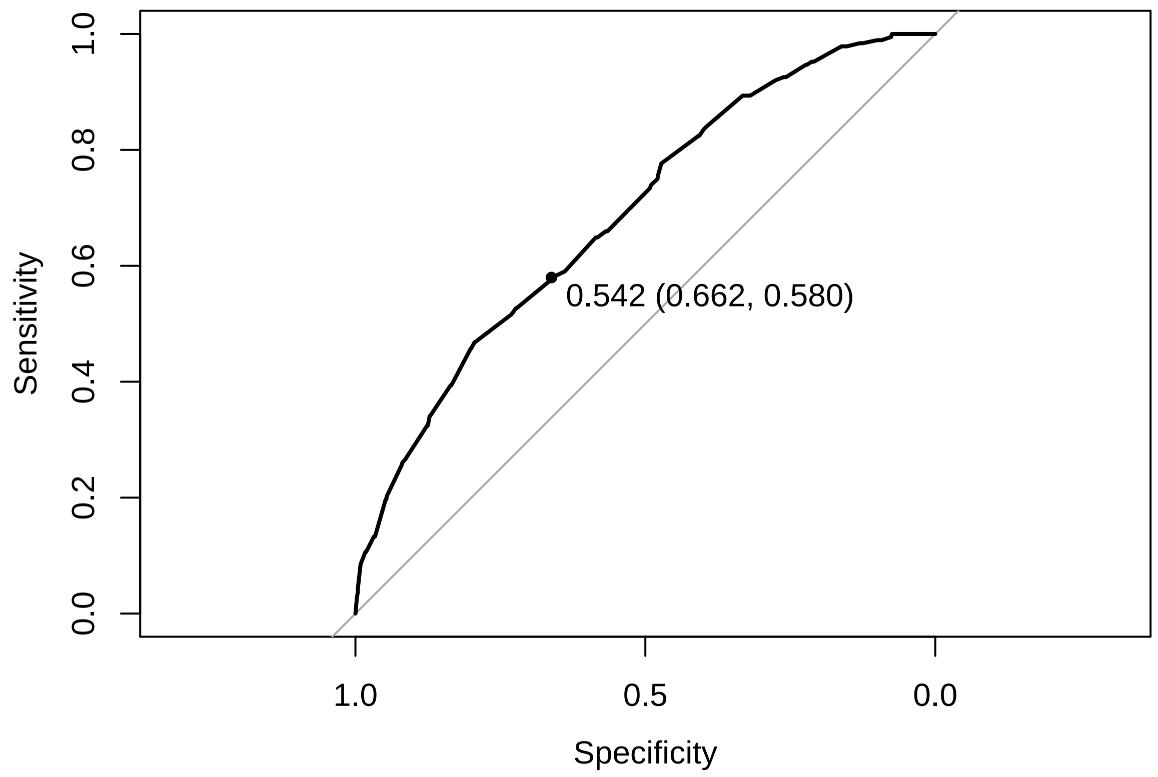
```
##
##  no yes
## 752 564
```

Set up train control for all machine learning models

```
over_sampling <- trainControl(method = "cv",
                              number = 10,
                              verboseIter = F, # don't print iterations
                              sampling = "smote", # over-sampling technique from chawla2002smote
                              savePredictions = "final",
                              classProbs=TRUE,
                              summaryFunction = twoClassSummary)
```

KNN

```
set.seed(2019)

knn_fit <- train(event ~ ., training,
                method = "knn",
                preProcess = c("center","scale"),
                #metric = "ROC",
                tuneLength = 15,
                trControl = over_sampling)

# Training
knn_train <- predict(knn_fit, type = "prob")

# Quick look at the ROC - caret doesn't check probability threshold I think
# Do it by hand for now, can write a function later

knn_train_roc <- roc(training$event, knn_train$yes)

# this shows the roc curve
plot(knn_train_roc, print.thres = "best", print.thres.best.method = "closest.topleft")
```



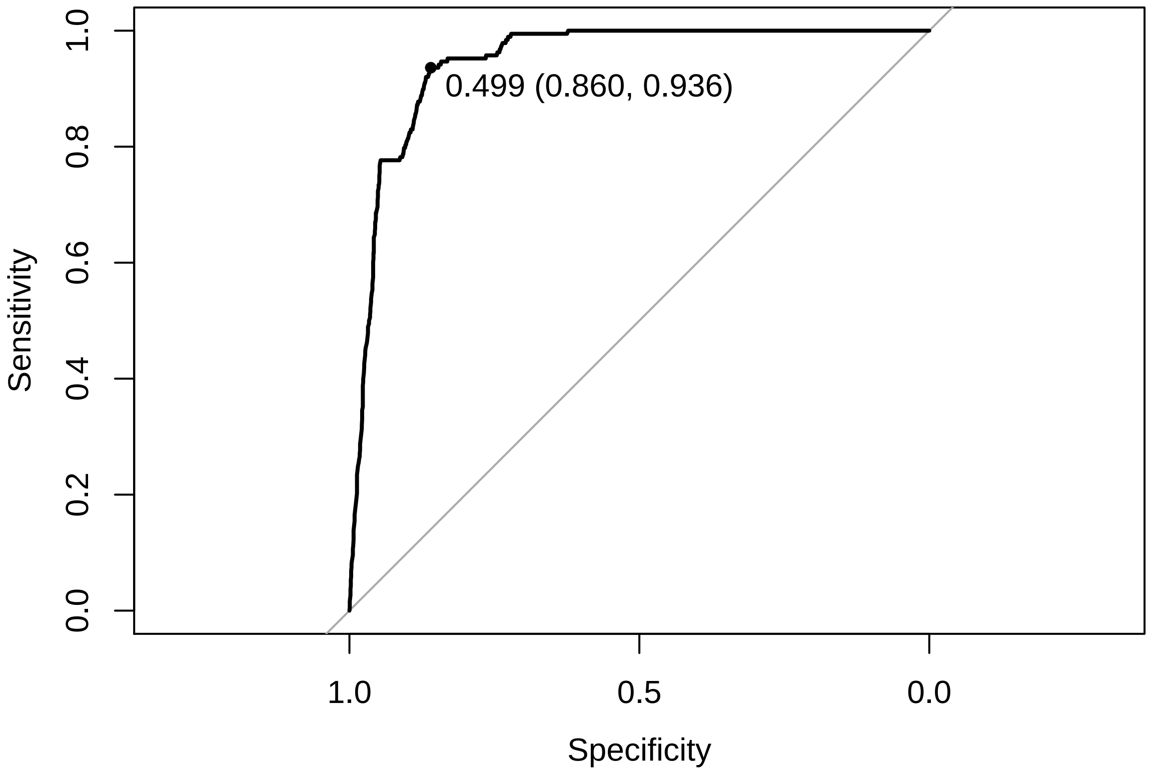
```
# look for the best top-left cutoff
knn_train_coords <- coords(knn_train_roc, "best", best.method = "closest.topleft", ret = c("threshold",

knn_train_labels <- training %>%
  mutate(predicted.value = factor(ifelse(knn_train[,2] <= knn_train_coords$threshold, "no", "yes"), leve

knn_train_error <- calc_error_rate(predicted.value = knn_train_labels$predicted.value, true.value = trai

# Test
knn_test <- predict(knn_fit, newdata = test, type = "prob")

knn_test_labels = test %>%
  mutate(predicted.value = factor(ifelse(knn_test[,2] <= knn_train_coords$threshold, "no", "yes"), level

knn_test_error <- calc_error_rate(predicted.value = knn_test_labels$predicted.value, true.value = test$

knn_test_roc <- roc(test$event, knn_test$yes)

knn_test_coords <- coords(knn_test_roc, "best", best.method = "closest.topleft", ret = c("threshold", "

# write down the training and test errors of the probit model
records[2,] <- c(knn_train_error, knn_test_error)
```

Random forest

```
set.seed(2019)

rf_fit <- train(event ~ ., training,
                method = "ranger",
                tuneLength = 15,
                #metric = "ROC",
                trControl = over_sampling)
```

```
## note: only 14 unique complexity parameters in default grid. Truncating the grid to 14 .
```

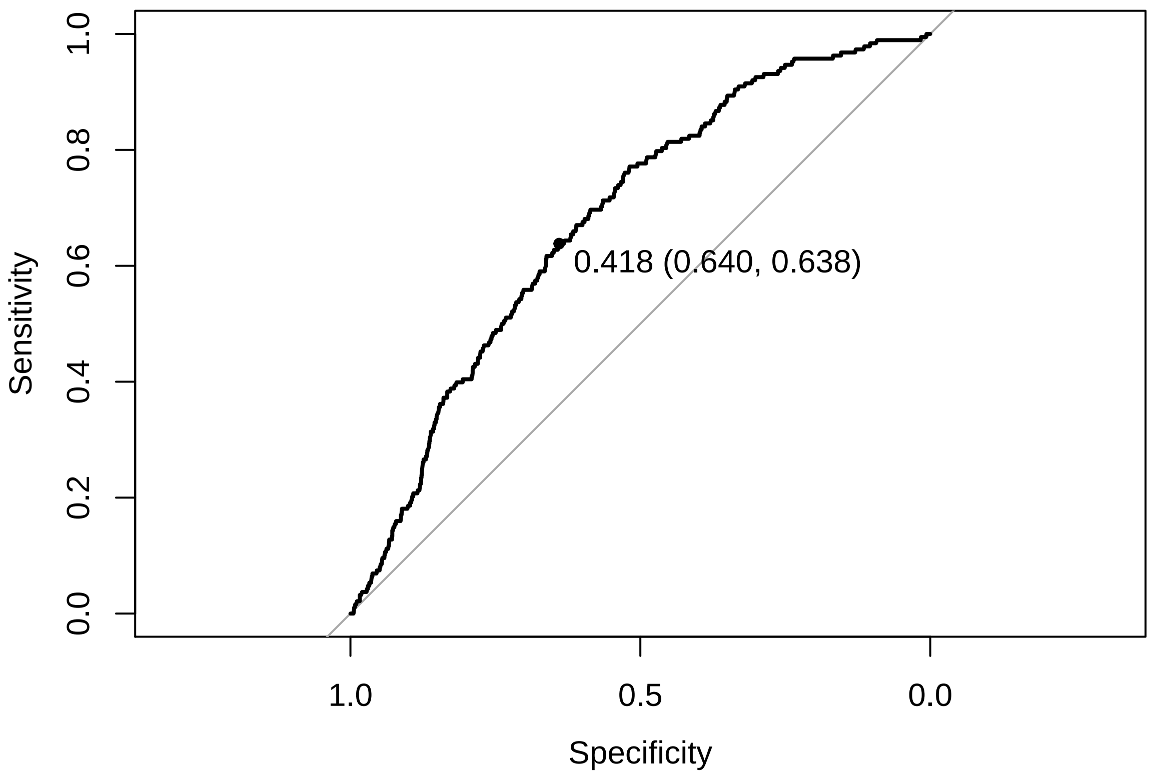
```
# Training
rf_train <- predict(rf_fit, training, type = "prob")

# Quick look at the ROC - caret doesn't check probability threshold I think
# Do it by hand for now, can write a function later

rf_train_roc <- roc(training$event, rf_train$yes)

# this shows the roc curve
plot(rf_train_roc, print.thres = "best", print.thres.best.method = "closest.topleft")
```



```
# look for the best top-left cutoff
rf_train_coords <- coords(rf_train_roc, "best", best.method = "closest.topleft", ret = c("threshold", "

rf_train_labels <- training %>%
  mutate(predicted.value = factor(ifelse(rf_train[,2] <= rf_train_coords$threshold, "no", "yes"), level

rf_train_error <- calc_error_rate(predicted.value = rf_train_labels$predicted.value, true.value = train

# Test
rf_test <- predict(rf_fit, newdata = test, type = "prob")

rf_test_labels = test %>%
  mutate(predicted.value = factor(ifelse(rf_test[,2] <= rf_train_coords$threshold, "no", "yes"), levels
rf_test_error <- calc_error_rate(predicted.value = rf_test_labels$predicted.value, true.value = test$eve

rf_test_roc <- roc(test$event, rf_test$yes)

rf_test_coords <- coords(rf_test_roc, "best", best.method = "closest.topleft", ret = c("threshold", "acc

# write down the training and test errors of the probit model
records[3,] <- c(rf_train_error, rf_test_error)
```

SVM

```
set.seed(2019)

svm_fit <- train(event ~ ., training,
                method = "svmLinear",
                preProcess = c("center","scale"),
                tuneLength = 15,
                #metric = "ROC",
                trControl = over_sampling)

# Training
svm_train <- predict(svm_fit, type = "prob")

# Quick look at the ROC - caret doesn't check probability threshold I think
# Do it by hand for now, can write a function later

svm_train_roc <- roc(training$event, svm_train$yes)

# this shows the roc curve
plot(svm_train_roc, print.thres = "best", print.thres.best.method = "closest.topleft")
```



```
# look for the best top-left cutoff
svm_train_coords <- coords(svm_train_roc, "best", best.method = "closest.topleft", ret = c("threshold",

svm_train_labels <- training %>%
  mutate(predicted.value = factor(ifelse(svm_train[,2] <= svm_train_coords$threshold, "no", "yes"), leve

svm_train_error <- calc_error_rate(predicted.value = svm_train_labels$predicted.value, true.value = trai

# Test
svm_test <- predict(svm_fit, newdata = test, type = "prob")

svm_test_labels = test %>%
  mutate(predicted.value = factor(ifelse(svm_test[,2] <= svm_train_coords$threshold, "no", "yes"), level

svm_test_error <- calc_error_rate(predicted.value = svm_test_labels$predicted.value, true.value = test$

svm_test_roc <- roc(test$event, svm_test$yes)

svm_test_coords <- coords(svm_test_roc, "best", best.method = "closest.topleft", ret = c("threshold", "

svm_test_error <- calc_error_rate(predicted.value = svm_test_labels$predicted.value, true.value = test$

# write down the training and test errors of the probit model
records[4,] <- c(svm_train_error, svm_test_error)
```

Model metrics

Accuracy

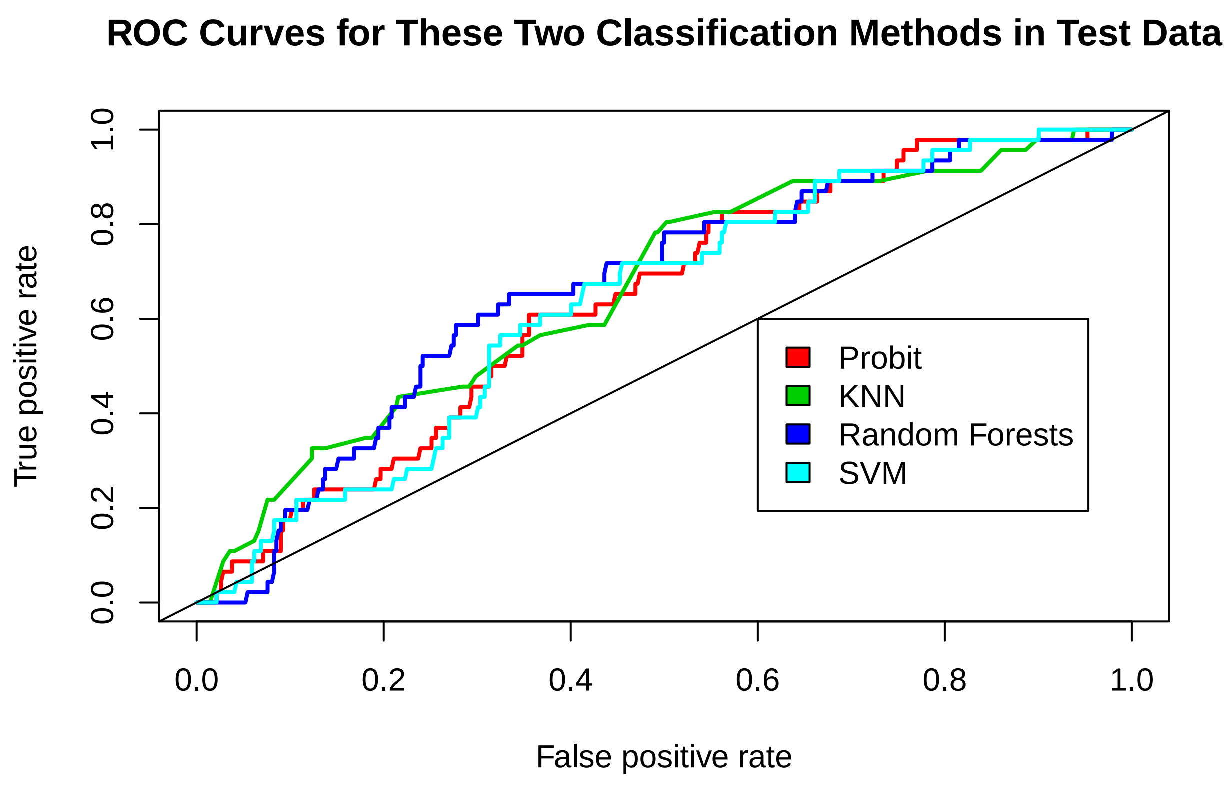
I won't read too much into it as it's a imbalanced class problem.

```
##                  train.error test.error
## Probit             0.3649441  0.3867521
## KNN                0.3462973  0.3589744
## Random Forests     0.1326585  0.2393162
## SVM                0.3601492  0.3888889
```

ROC curves

This is more important.

```r
#ROC on test data

# probit
pred_prob <- prediction(prob_test, test$event)
performance_prob <- performance(pred_prob, measure = "tpr", x.measure="fpr")

# knn
pred_knn <- prediction(knn_test[,2], test$event)
performance_knn <- performance(pred_knn, measure = "tpr", x.measure="fpr")

# rf
pred_rf <- prediction(rf_test[,2], test$event)
performance_rf <- performance(pred_rf, measure = "tpr", x.measure="fpr")

# svm
pred_svm <- prediction(svm_test[,2], test$event)
performance_svm <- performance(pred_svm, measure = "tpr", x.measure="fpr")

#plot
plot(performance_prob, col = 2, lwd = 2, main = "ROC Curves for These Two Classification Methods in Tes

legend(0.6, 0.6, c("Probit", "KNN", "Random Forests", "SVM"), 2:5)

#others
plot(performance_knn,col=3,lwd=2,add=TRUE)
plot(performance_rf,col=4,lwd=2,add=TRUE)
plot(performance_svm,col=5,lwd=2,add=TRUE)

abline(0,1)
```



AUC

Random forest and KNN stand out but not by much.

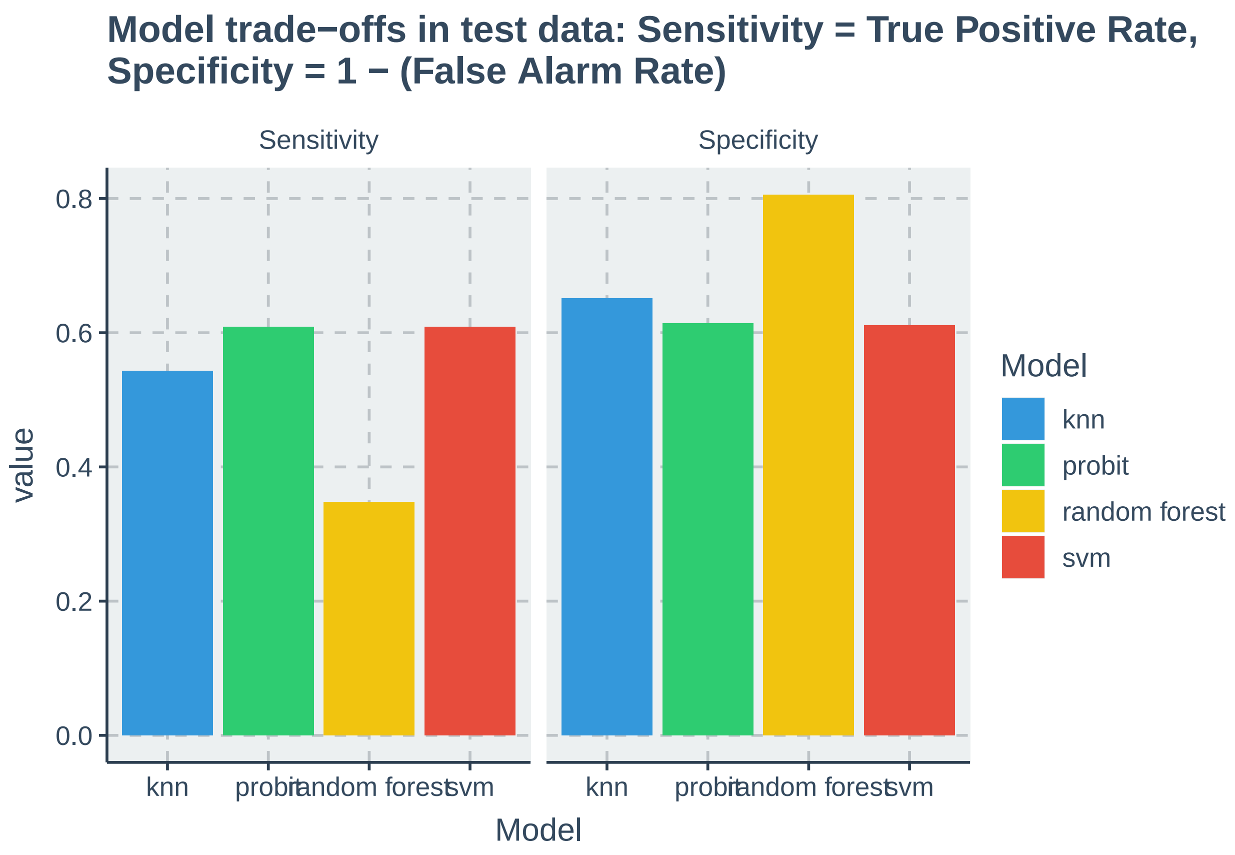
```
##                      AUC
## Probit         0.6380332
## KNN            0.6628632
## Random Forests 0.6661859
## SVM            0.6364362
```

Confusion Matrix

Trade offs between sensitivity and specificity

Looks like SVM doesn't bring much to the table…

```
# switching it because the way our table is made
sensitivity <- lapply(confusion, function(x) {data.frame("Sensitivity" = as.numeric(x$byClass[2]))}) %>%
specificity <- lapply(confusion, function(x) {data.frame("Specificity" = as.numeric(x$byClass[1]))}) %>%

main_metrics <- data.frame("Model" = c("probit", "knn", "random forest", "svm"),
                           sensitivity, specificity)

main_metrics_plot <- main_metrics %>% melt(id.var = "Model") %>%
  ggplot(aes(x = Model, y = value)) +
  geom_col(aes(fill = Model)) +
  facet_wrap(~variable) +
  labs(title = "Model trade-offs in test data: Sensitivity = True Positive Rate,\nSpecificity = 1 - (Fa

main_metrics_plot
```



Probit

```
##                       Reference
## Prediction             Actual = no Actual = yes
##    Prediction = no             259           18
##    Prediction = yes            163           28
```

KNN

```
##                       Reference
## Prediction             Actual = no Actual = yes
##    Prediction = no             275           21
##    Prediction = yes            147           25
```

Random Forest

```
##                       Reference
## Prediction             Actual = no Actual = yes
##    Prediction = no             340           30
##    Prediction = yes             82           16
```

SVM

```
##                       Reference
## Prediction             Actual = no Actual = yes
##    Prediction = no             258           18
##    Prediction = yes            164           28
```

References

Chawla, Nitesh V, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. “SMOTE: Synthetic Minority over-Sampling Technique.” *Journal of Artificial Intelligence Research* 16: 321–57.

Frankel, Jeffrey A, and Andrew K Rose. 1996. “Currency Crashes in Emerging Markets: An Empirical Treatment.” *Journal of International Economics* 41 (3-4). Elsevier: 351–66.