Deep Learning

Jameson Watts, Ph.D.

04/19/2020

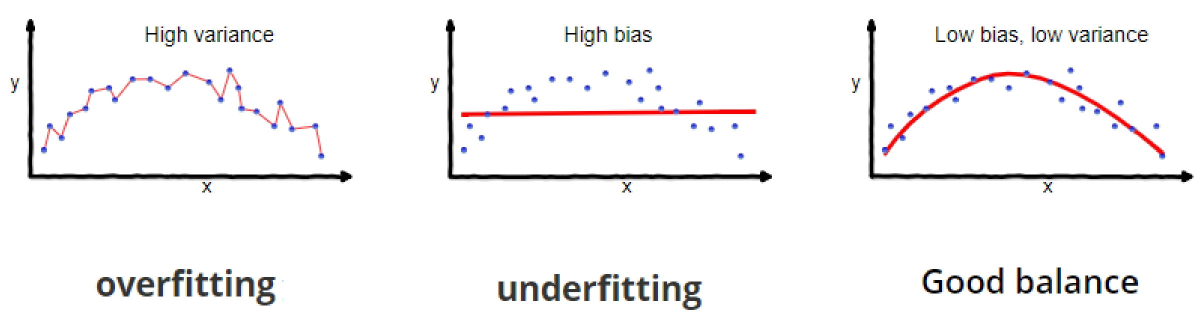
Agenda

1. Q&A with CravenSpeed
2. Anatomy of a Technical Presentation
3. Quick Review of Bagging and Boosting
4. Deep Learning Concepts
5. Dinner
6. Neural net implementation
7. Groupwork

Review of Bagging and Boosting

The goal is to decrease the variance (bagging) or bias (boosting) in our models.

- Step 1: producing a distribution of simple ML models on subsets of the original data.
- Step 2: combine the distribution into one "aggregated" model.



Dinner and (virtual) high fives

Neural Net implementations

Setup

```
knitr::opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE)
library(tidymodels)
library(tidyverse)
bank <- read_rds("../resources/BankChurners.rds") %>%
```

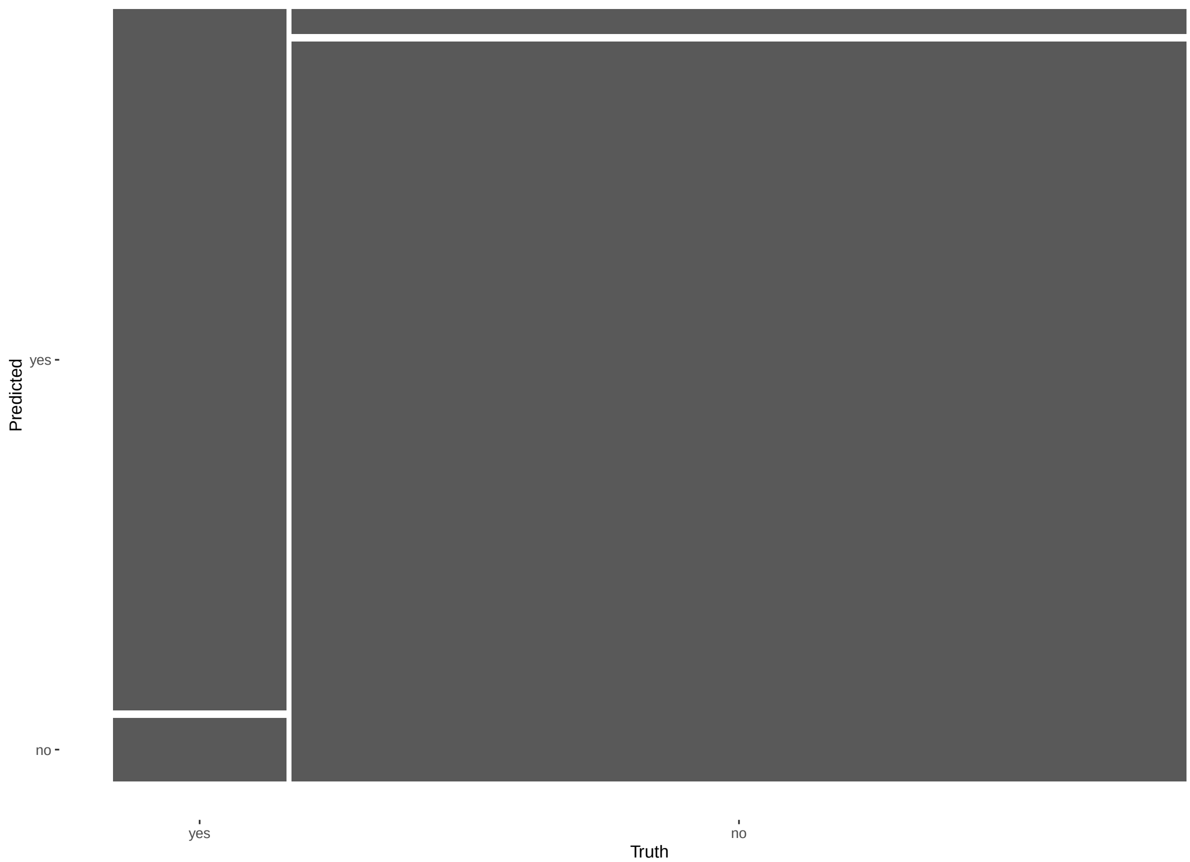
```r
  mutate(Churn = as_factor(Churn)) %>%
  mutate(Churn = fct_relevel(Churn, "yes","no"))

set.seed(504)
data_split <- initial_split(bank, prop = 3/4)

bank_train <- training(data_split)
bank_test  <- testing(data_split)
```

Set a baseline with extreme gradient boosting

```r
bank_rec <-
  recipe(Churn ~ ., data = bank_train) %>%
  step_BoxCox(all_numeric()) %>%
  step_normalize(all_numeric()) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%  # dummy variables for all factor/character columns ex
  step_zv(all_predictors()) %>%  # remove all zero variance predictors (i.e. low frequency dummies)
  step_upsample(Churn)

xgb_spec <-
   boost_tree() %>%
   set_engine("xgboost") %>%
   set_mode("classification")

bank_wflow <-
  workflow() %>%
  add_model(xgb_spec) %>%
  add_recipe(bank_rec)

bank_fit <- ## fit the model
  bank_wflow %>%
  fit(data = bank_train)
```

```
## [16:58:17] WARNING: amalgamation/../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evalu
```

```r
cm <- predict(bank_fit, bank_test) %>%
  bind_cols(bank_test %>% select(Churn))  %>%
  conf_mat(truth = Churn, .pred_class)

cm %>% autoplot()
```



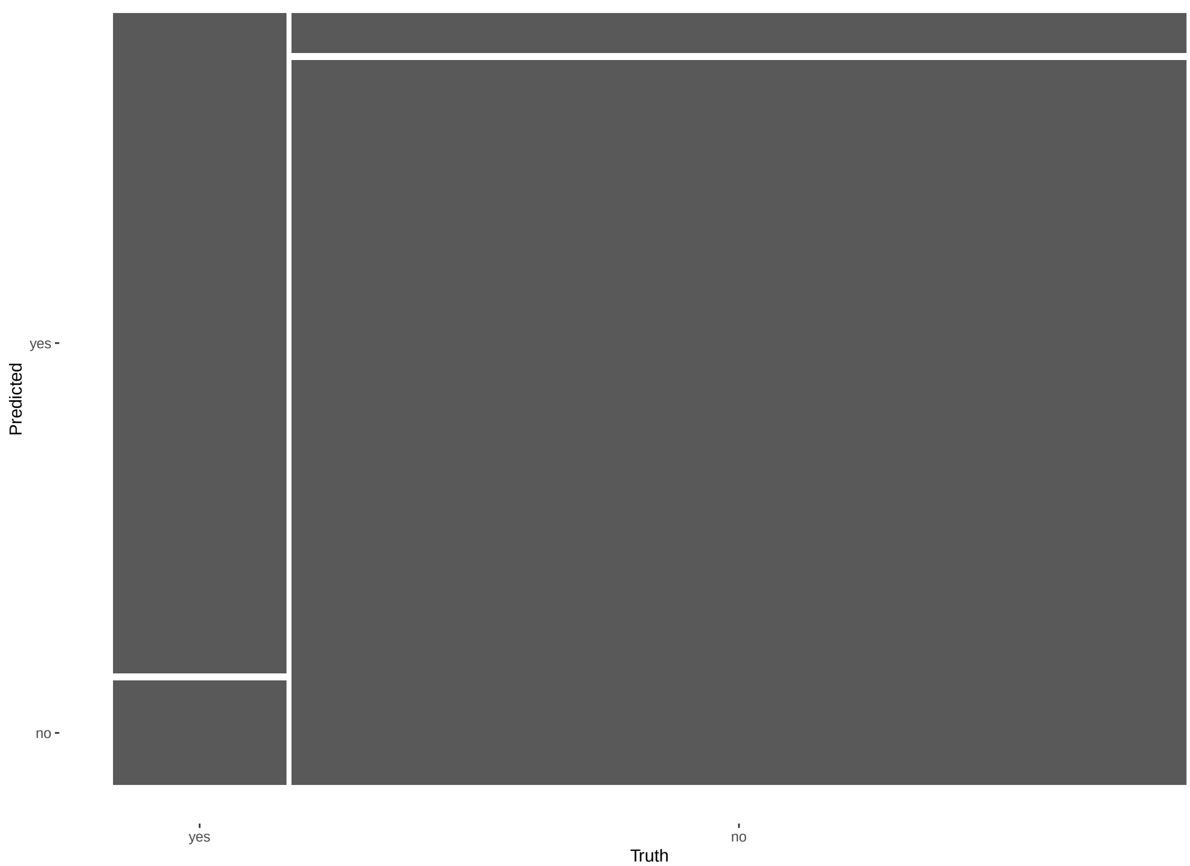

```
cm %>% summary()
```

.metric	.estimator	.estimate
accuracy	binary	0.9593046
kap	binary	0.8553678
sens	binary	0.9172749
spec	binary	0.9674528
ppv	binary	0.8452915
npv	binary	0.9836930
mcc	binary	0.8564027
j_index	binary	0.8847278
bal_accuracy	binary	0.9423639
detection_prevalence	binary	0.1762149
precision	binary	0.8452915
recall	binary	0.9172749
f_meas	binary	0.8798133

Compare with Neural Net

```
nnet_spec <-
  mlp(hidden_units = 11) %>%
  set_engine("nnet") %>%
  set_mode("classification")
```

```r
bank_wflow <-
  workflow() %>%
  add_model(nnet_spec) %>%
  add_recipe(bank_rec)

bank_fit <- ## fit the model
  bank_wflow %>%
  fit(data = bank_train)

cm <- predict(bank_fit, bank_test) %>%
  bind_cols(bank_test %>% select(Churn))  %>%
  conf_mat(truth = Churn, .pred_class)

cm %>% autoplot()
```




```r
cm %>% summary()
```

.metric	.estimator	.estimate
accuracy	binary	0.9344133
kap	binary	0.7710284
sens	binary	0.8637470
spec	binary	0.9481132
ppv	binary	0.7634409
npv	binary	0.9728945

.metric	.estimator	.estimate
mcc	binary	0.7731761
j_index	binary	0.8118602
bal_accuracy	binary	0.9059301
detection_prevalence	binary	0.1837218
precision	binary	0.7634409
recall	binary	0.8637470
f_meas	binary	0.8105023

Let's do some tuning

```
mlp_spec <-
  mlp(hidden_units = tune(), penalty = tune(), epochs = tune()) %>%
  set_engine("nnet", trace = 0) %>%
  set_mode("classification")


mlp_param <- parameters(mlp_spec)
mlp_param %>% pull_dials_object("hidden_units")
```

```
## # Hidden Units (quantitative)
## Range: [1, 10]
```

```
mlp_param %>% pull_dials_object("penalty")
```

```
## Amount of Regularization (quantitative)
## Transformer:  log-10
## Range (transformed scale): [-10, 0]
```

```
mlp_param %>% pull_dials_object("epochs")
```

```
## # Epochs (quantitative)
## Range: [10, 1000]
```

defining your own grid

```
crossing(
  hidden_units = 1:3,
  penalty = c(0.0, 0.1),
  epochs = c(100, 200)
)
```

hidden_units	penalty	epochs
1	0.0	100
1	0.0	200
1	0.1	100
1	0.1	200
2	0.0	100
2	0.0	200
2	0.1	100
2	0.1	200
3	0.0	100
3	0.0	200

hidden_units	penalty	epochs
3	0.1	100
3	0.1	200

using grid_regular

```
grid_regular(mlp_param, levels = 2)
```

hidden_units	penalty	epochs
1	0	10
10	0	10
1	1	10
10	1	10
1	0	1000
10	0	1000
1	1	1000
10	1	1000

setting different levels

```
grid_regular(mlp_param, levels = c(hidden_units = 3, penalty = 2, epochs = 2))
```

hidden_units	penalty	epochs
1	0	10
5	0	10
10	0	10
1	1	10
5	1	10
10	1	10
1	0	1000
5	0	1000
10	0	1000
1	1	1000
5	1	1000
10	1	1000

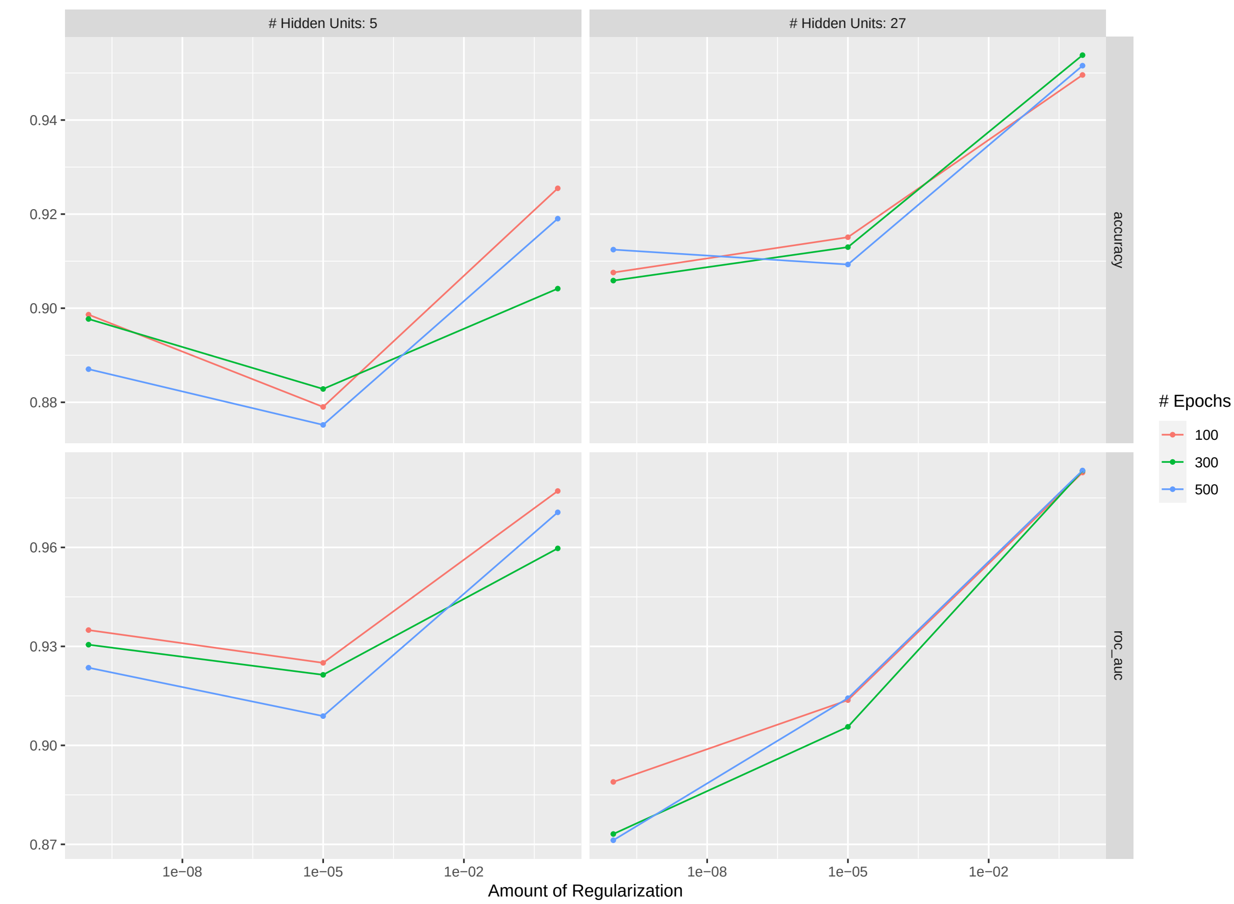
let's change the defaults

```
mlp_wflow <-
  workflow() %>%
  add_model(mlp_spec) %>%
  add_recipe(bank_rec)

mlp_param <-
  mlp_wflow %>%
  parameters() %>%
  update(
    epochs = epochs(c(100, 500)),
    hidden_units = hidden_units(c(5, 50))
```

```
  )
```

run all models across the grid

```
set.seed(504)
folds <- vfold_cv(bank_train, v = 2)

mlp_reg_tune <-
  mlp_wflow %>%
  tune_grid(
    folds,
    grid = mlp_param %>% grid_regular(levels = 3)
  )

autoplot(mlp_reg_tune)
```



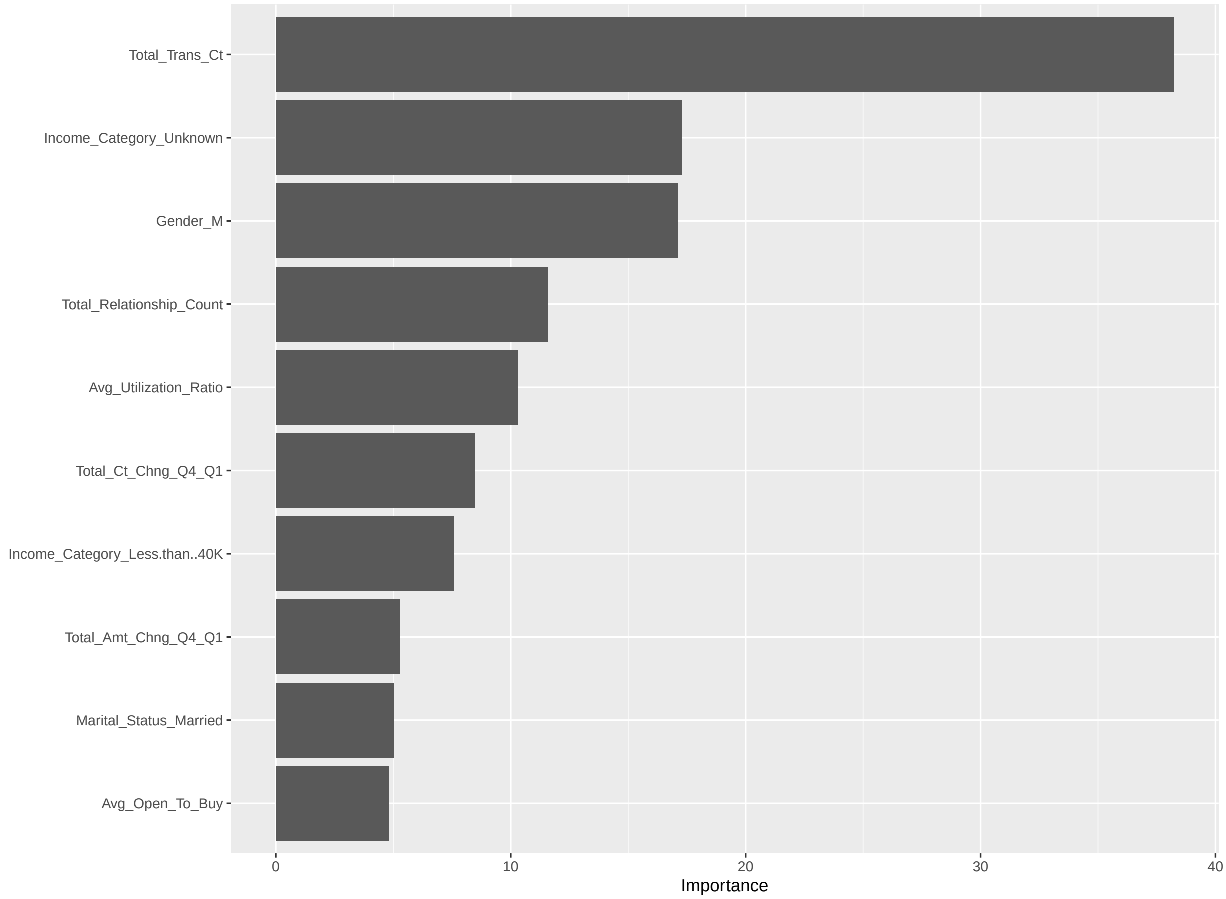

```
show_best(mlp_reg_tune) %>% select(-.estimator)
```

hidden_units	penalty	epochs	.metric	mean	n	std_err	.config
27	1	500	roc_auc	0.9833093	2	0.0014108	Preprocessor1_Model26
27	1	300	roc_auc	0.9832088	2	0.0008288	Preprocessor1_Model17
27	1	100	roc_auc	0.9826945	2	0.0004908	Preprocessor1_Model08
5	1	100	roc_auc	0.9770975	2	0.0022923	Preprocessor1_Model07

hidden_units	penalty	epochs	.metric	mean	n	std_err	.config
5	1	500	roc_auc	0.9706361	2	0.0001105	Preprocessor1_Model25

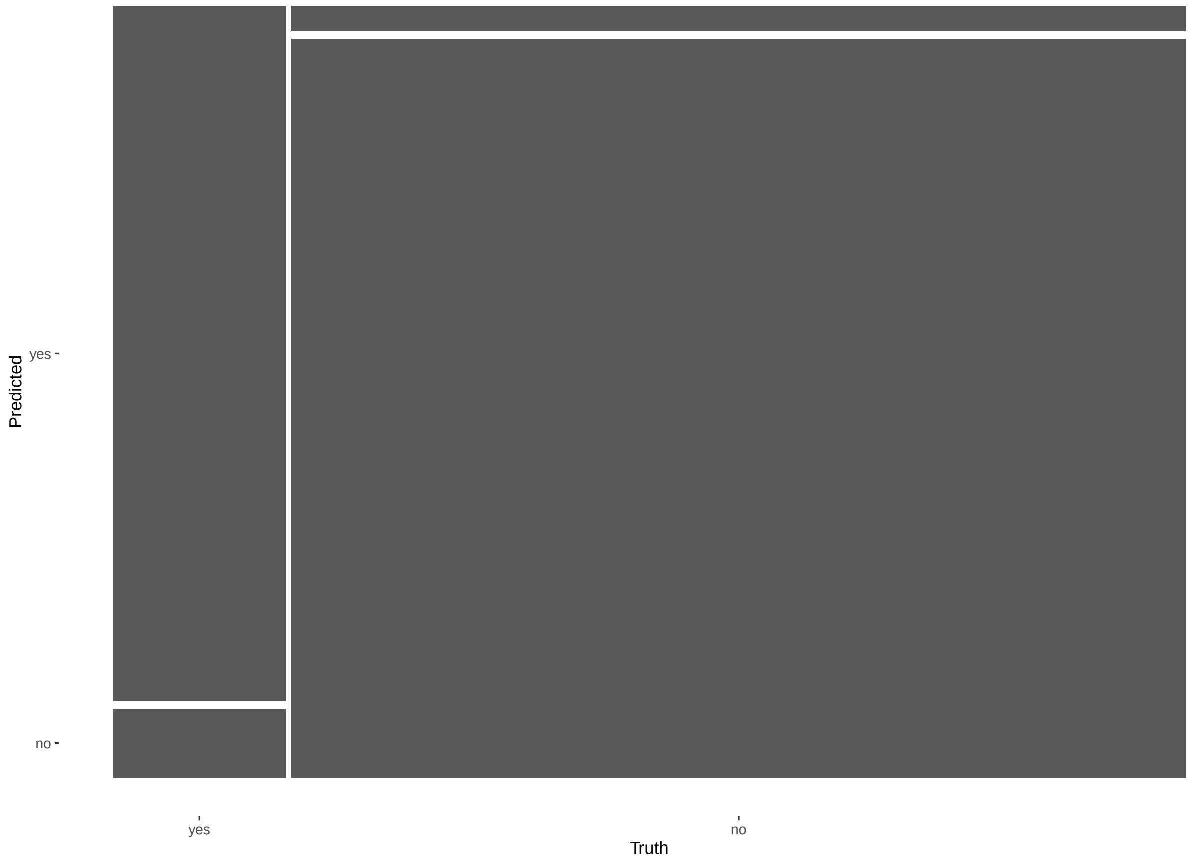
Finalize, fit and pull our optimized model

```
best_net <- mlp_reg_tune %>%
  select_best("roc_auc")

final_wflow <-
  mlp_wflow %>%
  finalize_workflow(best_net)

bank_fit <-
  final_wflow %>%
  fit(data = bank_train)

library(vip)

bank_fit %>%
  pull_workflow_fit() %>%
  vip()
```



Let's see how it does out of sample

```
cm <- predict(bank_fit, bank_test) %>%
  bind_cols(bank_test %>% select(Churn))  %>%
  conf_mat(truth = Churn, .pred_class)

cm %>% autoplot()
```




```
cm %>% summary()
```

.metric	.estimator	.estimate
accuracy	binary	0.9577242
kap	binary	0.8494656
sens	binary	0.9099757
spec	binary	0.9669811
ppv	binary	0.8423423
npv	binary	0.9822712
mcc	binary	0.8503825
j_index	binary	0.8769568
bal_accuracy	binary	0.9384784
detection_prevalence	binary	0.1754247
precision	binary	0.8423423
recall	binary	0.9099757
f_meas	binary	0.8748538

Other resources

https://srdas.github.io/DLBook/