Applications

Of Data Science

Topics in Classification

Applications of Data Science - Class 13

Giora Simchoni

gsimchoni@gmail.com and add #dsapps in subject

Stat. and OR Department, TAU

2021-01-19

Topics in Classification

Life isn't perfect

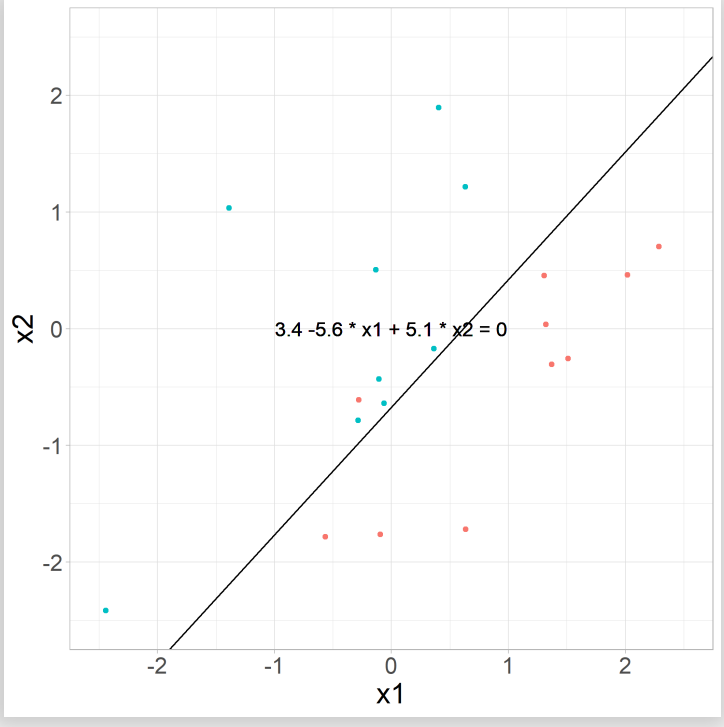
Let's tackle just a few issues:

- Not enough labelled data and data labeling is expensive
- Imbalanced Classes

Active Learning

Got Data?

```
n <- 20
x1 <- rnorm(n, 0, 1); x2 <- rnorm(n, 0, 1)
t <- 2 - 4 * x1 + 3 * x2
y <- rbinom(n, 1, 1 / (1 + exp(-t)))
glm_mod <- glm(y ~ x1 + x2, family = "binomial")
```



Want more?

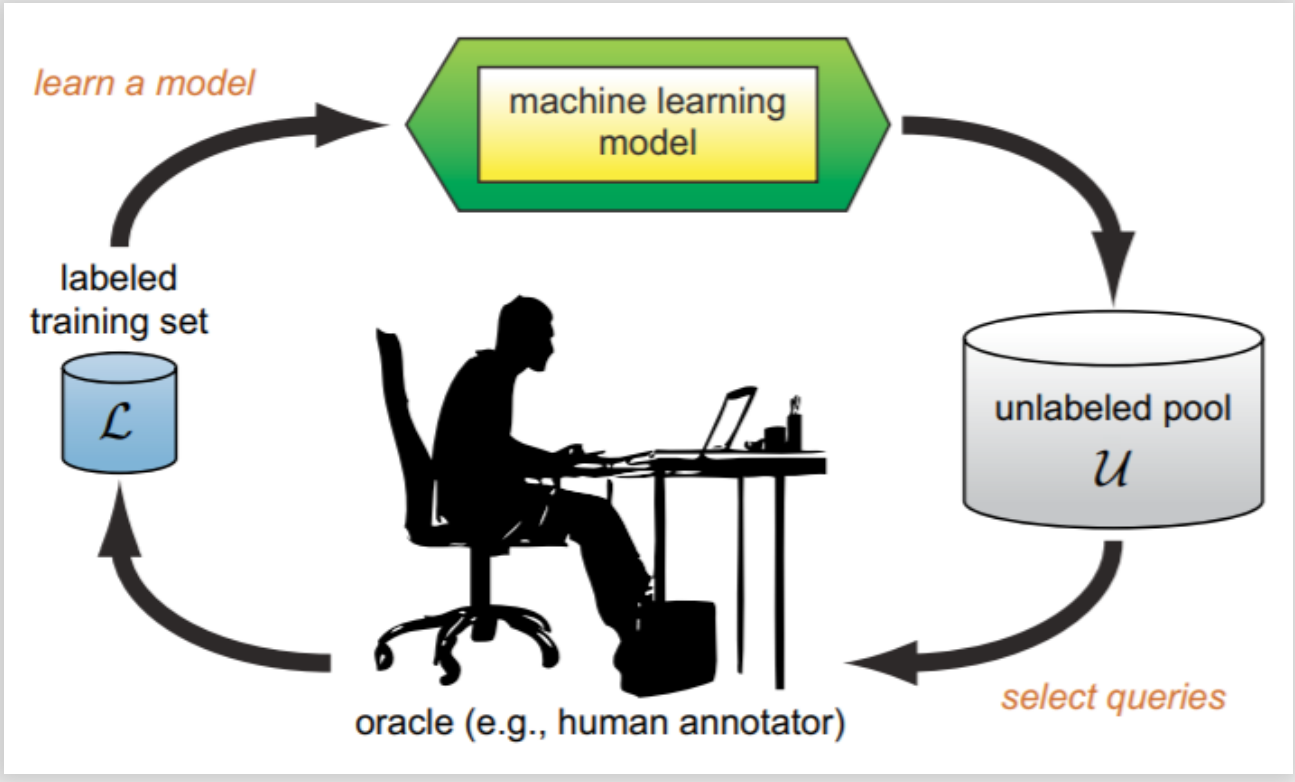
> The key idea behind *active learning* is that a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns. An active learner may pose *queries*, usually in the form of unlabeled data instances to be labeled by an *oracle* (e.g., a human annotator). Active learning is well-motivated in many modern machine learning problems, where unlabeled data may be abundant or easily obtained, but labels are difficult, time-consuming, or expensive to obtain.

(Settles, 2010)

> You want data? Well data costs!

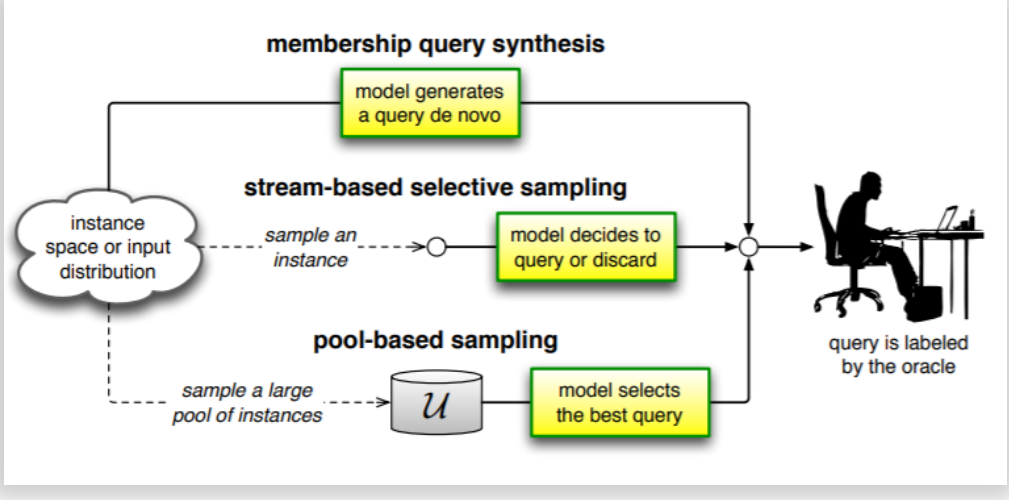
(No one, ever)

Where this is going



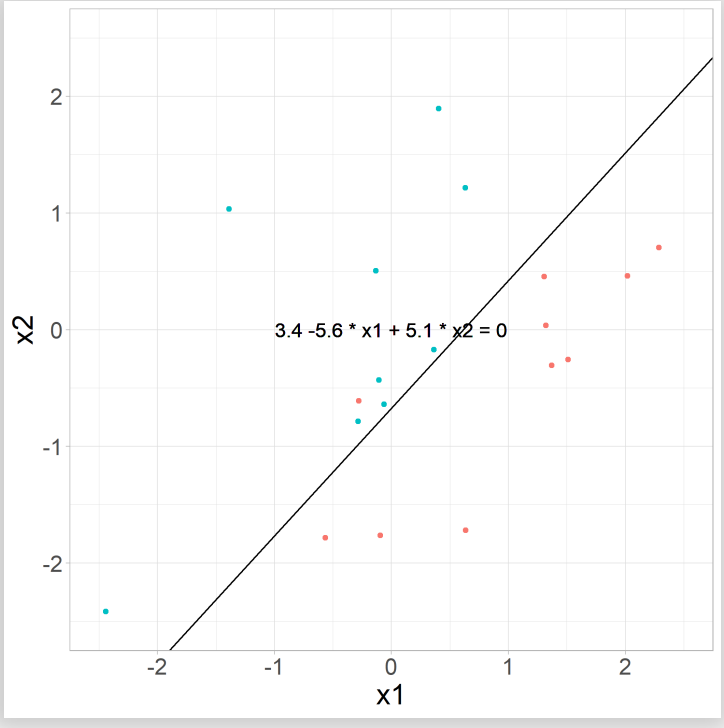
Active Learning Scenarios

1. **Membership Query Synthesis**: You get to choose which (maybe theoretical) points you'd want y labelled for.
2. **Stream-Based Selective Sampling**: You get 1 point at a time and decide which ones you'd like to query and which to discard.
3. **Pool-Based Sampling**: You have a large collecetion of unlabelled points at your disposal, you need to send the "best ones" for labelling



Uncertainty Sampling

For a 2-class dataset, the observations your model is most uncertain of are...



Uncertainty Sampling Measures

Let $\hat{y}_i$ be the predicted classes with ith highest score (probability), for observations x under some model θ.

So $\hat{y}_1 = \arg\max P_\theta(y|x)$ are the actual predicted classes, $\hat{y}_2$ are the second choices, etc.

- Least Confidence: Choose those observations for which $P_\theta(\hat{y}_1|x)$ is smallest:

$$x^*_{LC} = \arg\min P_\theta(\hat{y}_1|x)$$

> For a 2-class balanced dataset, this means...

- Margin Sampling: Choose those observations for which the margin between the two highest scores is smallest:

$$x^*_M = \arg\min P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)$$

For a 2-class balanced dataset, this means...

- Entropy: Choose the observations for which entropy is highest:

$$x^*_H = \arg\max -\sum_i P_\theta(\hat{y}_i|x) \log[P_\theta(\hat{y}_i|x)]$$

We will talk more about entropy in Neural Networks, let's minimize negative entropy.

For a 2-class balanced dataset, this means...

Example: The `spotify_songs` data from HW3

```
spotify_songs <- read_csv('https://raw.githubusercontent.com/rforc

spotify_songs %>% count(playlist_genre)
```

```
## # A tibble: 6 x 2
##   playlist_genre     n
##   <chr>          <int>
## 1 edm             6043
## 2 latin           5155
## 3 pop             5507
## 4 r&b             5431
## 5 rap             5746
## 6 rock            4951
```

Let's try to classify the genre of a song!

We'll take only the 12 audio features as predictors, and choose each `track_id` once (remember each song appears a few times?):

```
library(tidymodels)

predictors <- 12:23

spotify_songs <- spotify_songs %>%
  group_by(track_id) %>%
  slice_sample(n = 1) %>%
  ungroup() %>%
  distinct(track_name, .keep_all = TRUE) %>%
  select(track_id, track_name, track_artist, playlist_genre, predi
  mutate(playlist_genre = recode(playlist_genre, "r&b" = "rnb"))

set.seed(76)
sptfy_split_obj <- spotify_songs %>%
  initial_split(prop = 0.8)
sptfy_tr <- training(sptfy_split_obj)
sptfy_te <- testing(sptfy_split_obj)
```

Plot twist! We only have 20 songs from each genre!

```
sptfy_tr_small <- sptfy_tr %>%
  group_by(playlist_genre) %>%
  slice_sample(n = 20) %>%
  ungroup()

sptfy_tr_small %>% count(playlist_genre)
```

```
## # A tibble: 6 x 2
##   playlist_genre     n
##   <chr>          <int>
## 1 edm               20
## 2 latin             20
## 3 pop               20
## 4 rap               20
## 5 rnb               20
## 6 rock              20
```

Muhaha!

We'll also have a pool of songs to query, `sptfy_tr_large`:

```
sptfy_tr_large <- sptfy_tr %>%
  anti_join(sptfy_tr_small, by = "track_id")
```

We `bake()` the 3 datasets with the small sample params recipe:

```
sptfy_rec <- recipe(playlist_genre ~ ., data = sptfy_tr_small) %>%
  update_role(track_id, track_name, track_artist,
              new_role = "id") %>%
  step_normalize(all_numeric(), -has_role("id")) %>%
  step_string2factor(playlist_genre) %>%
  prep(sptfy_tr_small, strings_as_factors = FALSE)

sptfy_tr_small <- juice(sptfy_rec)
sptfy_tr_large <- bake(sptfy_rec, new_data = sptfy_tr_large)
sptfy_te <- bake(sptfy_rec, new_data = sptfy_te)
```

Let's build a simple GBT model:

```
mod_spec <- boost_tree(mode = "classification", trees = 100) %>%
  set_engine("xgboost", eval_metric = "mlogloss")

mod_fit <- mod_spec %>%
  fit(playlist_genre ~ ., data = sptfy_tr_small %>%
        select(-track_id, -track_name, -track_artist))

mod_pred <- mod_fit %>%
  predict(new_data = sptfy_tr_large, type = "prob")

mod_pred
```

```
## # A tibble: 18,641 x 6
##    .pred_edm .pred_latin .pred_pop .pred_rap .pred_rnb .pred_rock
##        <dbl>       <dbl>     <dbl>     <dbl>     <dbl>      <dbl>
##  1   0.00584     0.0700    0.455     0.0115    0.320      0.137
##  2   0.00937     0.00305   0.00108   0.00127   0.00240    0.983
##  3   0.879       0.0547    0.0530    0.00402   0.00224    0.00673
##  4   0.785       0.0578    0.0530    0.0146    0.0843     0.00502
##  5   0.0553      0.198     0.200     0.205     0.186      0.155
##  6   0.00147     0.00447   0.0235    0.00335   0.947      0.0205
##  7   0.383       0.465     0.0421    0.0471    0.0517     0.0113
##  8   0.00190     0.185     0.00992   0.762     0.0281     0.0135
##  9   0.590       0.0594    0.0311    0.0977    0.213      0.00880
## 10   0.0306      0.616     0.142     0.108     0.0985     0.00516
## # ... with 18,631 more rows
```

Test accuracy?

```
mod_te_pred_class <- mod_fit %>%
      predict(new_data = sptfy_te) %>%
      bind_cols(sptfy_te)

mod_te_pred_class %>%
  accuracy(truth = playlist_genre, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.387
```

Remember this model was built on 120 of almost 19K available unique songs!

Test Recall and Precision:

```
mod_te_pred_class %>%
  group_by(playlist_genre) %>%
  accuracy(truth = playlist_genre, estimate = .pred_class) %>%
  select(playlist_genre, recall = .estimate) %>%
  bind_cols(
    mod_te_pred_class %>%
      group_by(.pred_class) %>%
      accuracy(truth = playlist_genre, estimate = .pred_class) %>%
      select(precision = .estimate)
  )
```

```
## # A tibble: 6 x 3
##   playlist_genre recall precision
##   <fct>           <dbl>     <dbl>
## 1 edm             0.586     0.625
## 2 latin           0.252     0.259
## 3 pop             0.274     0.243
## 4 rap             0.357     0.532
## 5 rnb             0.312     0.263
## 6 rock            0.536     0.429
```

Build a function which will take each row of predicted probs and return a list of 3 uncertainty metrics:

```
uncertainty_lc <- function(probs) {
  max(probs)
}

uncertainty_m <- function(probs) {
  o <- order(probs, decreasing = TRUE)
  probs[o[1]] - probs[o[2]]
}

uncertainty_h <- function(probs) {
  sum(probs * log(probs + 0.000001))
}

uncertainty <- function(...) {
  probs <- c(...)
  list(
    lc = uncertainty_lc(probs),
    margin = uncertainty_m(probs),
    entropy = uncertainty_h(probs)
  )
}
```

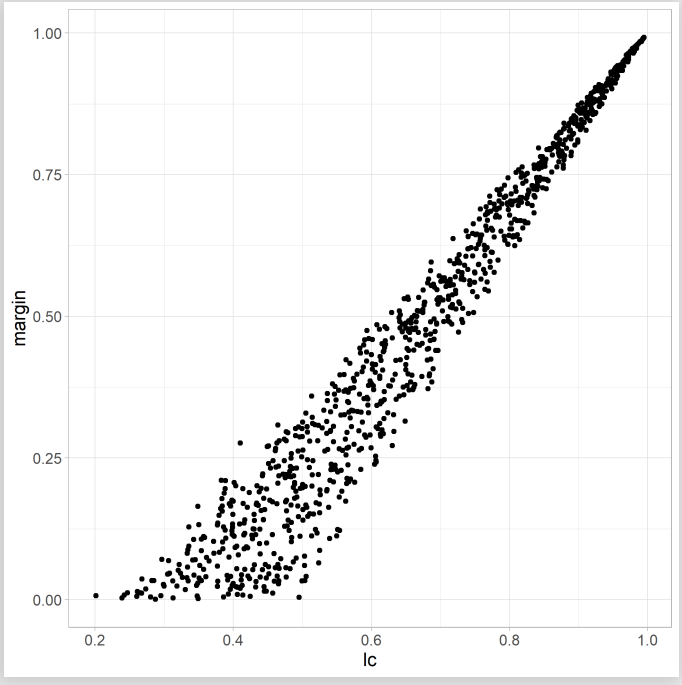
```
mod_unc <- mod_pred %>% pmap_dfr(uncertainty)

mod_unc
```

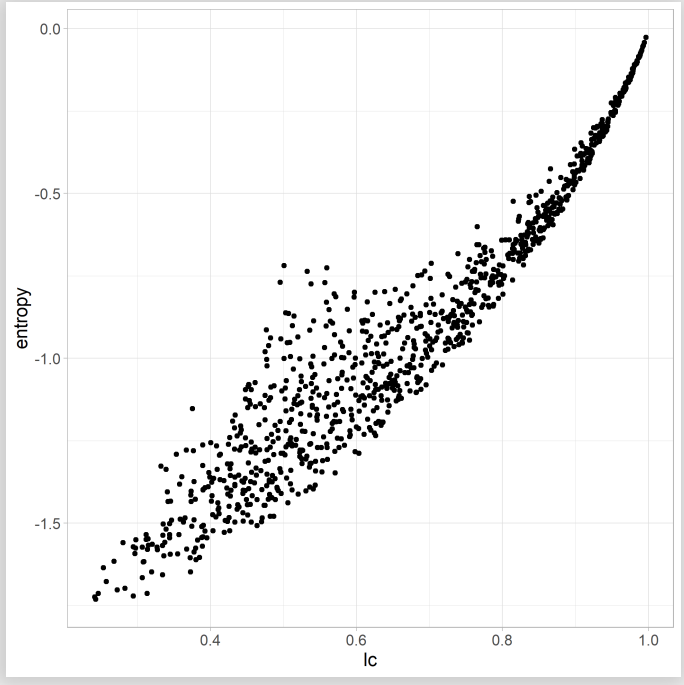
```
## # A tibble: 18,641 x 3
##       lc  margin entropy
##    <dbl>   <dbl>   <dbl>
##  1 0.455 0.135    -1.26
##  2 0.983 0.973    -0.109
##  3 0.879 0.824    -0.497
##  4 0.785 0.701    -0.807
##  5 0.205 0.00491  -1.73
##  6 0.947 0.923    -0.272
##  7 0.465 0.0814   -1.20
##  8 0.762 0.577    -0.736
##  9 0.590 0.377    -1.19
## 10 0.616 0.474    -1.18
## # ... with 18,631 more rows
```

Obviously these are correlated:

```
mod_unc %>% slice_sample(n = 1(
  ggplot(aes(lc, margin)) +
  geom_point() +
  theme_light() +
  theme(text =
          element_text(size = 1
```




```
mod_unc %>% slice_sample(n = 1(
  ggplot(aes(lc, entropy)) +
  geom_point() +
  theme_light() +
  theme(text =
          element_text(size = 1
```



Which are the top 10 songs in terms of each metric the model is most curious about?

```
sptfy_tr_large_with_unc <- sptfy_tr_large %>%
  bind_cols(mod_unc) %>%
  select(track_name, track_artist, playlist_genre, lc, margin, ent

sptfy_tr_large_with_unc %>%
  slice_min(lc, n = 10) %>%
  arrange(lc, track_name)
```

```
## # A tibble: 10 x 6
##    track_name                  track_artist  playlist_genre     lc  margin
##    <chr>                       <chr>         <fct>           <dbl>   <dbl>
##  1 REPEAT (feat. Pistol Pete~ Jaecy         rap             0.201 6.66e-3
##  2 The Greatest               Sia           pop             0.204 1.22e-2
##  3 Taxi Voy                   j mena        latin           0.205 1.79e-2
##  4 Poison                     Bell Biv DeV~ rnb             0.205 4.91e-3
##  5 Super Human                Tobi          rap             0.216 8.70e-3
##  6 Any Time, Any Place        Janet Jackson rnb             0.217 1.11e-2
##  7 goodbye                    updog         pop             0.222 1.86e-2
##  8 Purple Eyes (feat. Phoebe~ The Knocks    pop             0.228 8.44e-4
##  9 Save Me (feat. Eneli) (fe~ Mahmut Orhan  edm             0.231 8.99e-3
## 10 Oh My Darling Don't Cry    Run The Jewe~ pop             0.233 9.54e-3
```

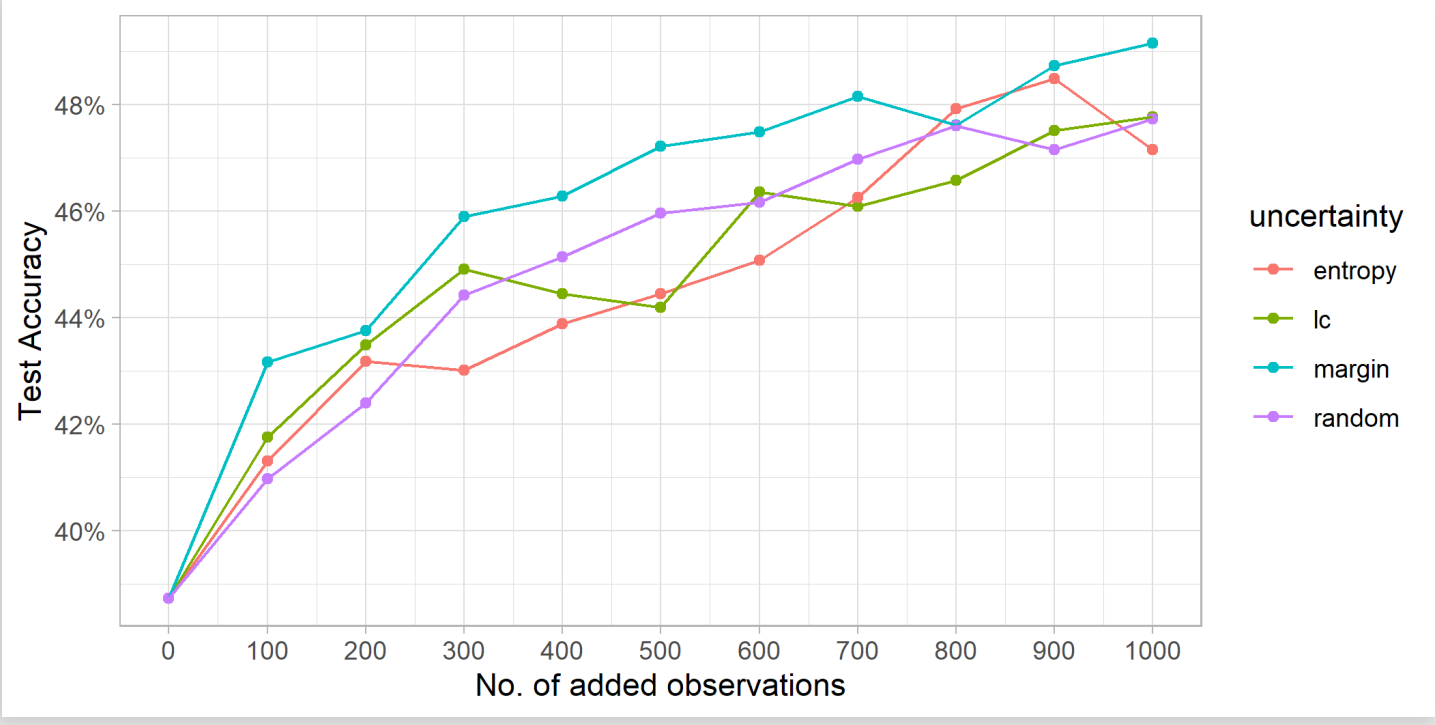
```
sptfy_tr_large_with_unc %>%
  slice_min(margin, n = 10) %>%
  arrange(margin, track_name)
```

```
## # A tibble: 10 x 6
##    track_name                    track_artist playlist_genre    lc   margin
##    <chr>                         <chr>        <fct>          <dbl>    <dbl>
##  1 Physical                      C'SAR        edm            0.440  2.93e-5
##  2 Last Child                    Aerosmith    rock           0.336  3.79e-5
##  3 Losing                        H.E.R.       rnb            0.371  8.41e-5
##  4 Saint-Tropez                  Post Malone  rap            0.420  8.82e-5
##  5 Black Magic Woman - Singl~ Santana      rock           0.438  1.38e-4
##  6 River                         Sal & Smit   edm            0.242  1.54e-4
##  7 The Octagon (Ready For Th~ Chillinit    rap            0.305  1.59e-4
##  8 Eazy-er Said Than Dunn        Eazy-E       rap            0.318  1.89e-4
##  9 Ella Quiere Beber - Remix     Anuel AA     latin          0.382  2.08e-4
## 10 Lord I Need You Now           Cynthia Jon~ rnb            0.329  2.15e-4
```

```
sptfy_tr_large_with_unc %>%
  slice_min(entropy, n = 10) %>%
  arrange(entropy, track_name)
```

```
## # A tibble: 10 x 6
##    track_name                    track_artist playlist_genre    lc  margin
##    <chr>                         <chr>        <fct>          <dbl>   <dbl>
##  1 The Greatest                  Sia          pop            0.204 0.0122
##  2 Taxi Voy                      j mena       latin          0.205 0.0179
##  3 REPEAT (feat. Pistol Pete ~ Jaecy        rap            0.201 0.00666
##  4 Super Human                   Tobi         rap            0.216 0.00870
##  5 goodbye                       updog        pop            0.222 0.0186
##  6 Any Time, Any Place           Janet Jacks~ rnb            0.217 0.0111
##  7 Long Away - Remastered 2011 Queen        rock           0.241 0.0580
##  8 Warrior Part 2                Lloyd Banks  rap            0.239 0.0268
##  9 Come Back To Me - filous R~ Urban Cone   pop            0.247 0.0671
## 10 Boss                          Jass Manak   rap            0.251 0.0598
```

So far it's only interesting. Will sending the observations our model is most curious about to the "oracle" prove to increase test accuracy better than random observations? See full code in slides Rmd files.



Query by Commity (QBC)

Similar to ensemble models, we have a committee of models:

$$C = \{\theta_1, \ldots, \theta_C\}$$

Which observations the commitee is most uncertain of? E.g.

$$x^*_{VE} = \arg\max -\sum_i \frac{V(\hat{y}_i|x)}{|C|} \log \frac{V(\hat{y}_i|x)}{|C|}$$

Where $V(\hat{y}_i|x)$ is the number of votes for $\hat{y}_i$.

How do you get a committee?

- Different models
- Bagging
- Same model, different subsets of features
- Same model, different params

Let's do 6 GBT models, each receiving 2 different consecutive features:

```
fit_sub_model <- function(i, tr, te) {
  mod_fit <- mod_spec %>%
    fit(playlist_genre ~ ., data = tr %>%
          select(playlist_genre, (2 + i * 2):(3 + i * 2)))

  mod_fit %>%
    predict(new_data = te)
}
mod_pred <- map_dfc(1:6, fit_sub_model,
                    tr = sptfy_tr_small, te = sptfy_tr_large)
mod_pred
```

```
## # A tibble: 18,641 x 6
##    .pred_class...1 .pred_class...2 .pred_class...3 .pred_class...4
##    <fct>           <fct>           <fct>           <fct>
##  1 latin           rock            rock            rock
##  2 rock            rock            rnb             latin
##  3 pop             rock            rnb             rock
##  4 pop             rnb             edm             rock
##  5 rap             latin           rap             pop
##  6 rock            pop             rock            rnb
##  7 latin           rnb             latin           rap
##  8 rock            rock            rap             latin
##  9 pop             latin           rock            pop
## 10 rnb             latin           pop             pop
```

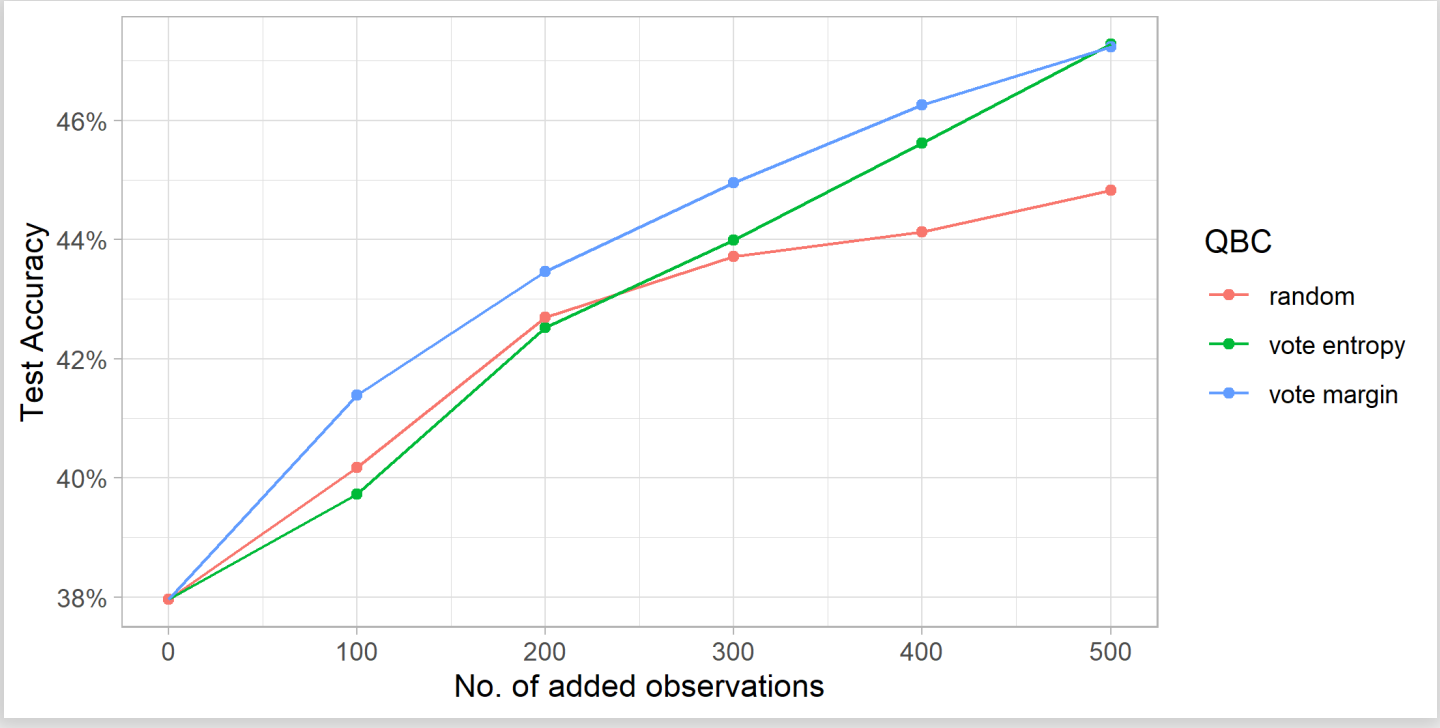
```
mod_qbc <- mod_pred %>%
  mutate(probs = pmap(
    select(., starts_with(".pred")),
    function(...) table(c(...)) / 6),
    vote_entropy = map_dbl(probs, uncertainty_h),
    vote_margin = map_dbl(probs, uncertainty_m))

sptfy_tr_large_with_qbc <- sptfy_tr_large %>%
  bind_cols(mod_qbc) %>%
  select(track_name, track_artist, playlist_genre,
         starts_with(".pred"), vote_entropy)

sptfy_tr_large_with_qbc %>%
  slice_min(vote_entropy, n = 10) %>%
  arrange(vote_entropy) %>%
  select(starts_with(".pred"))
```

```
## # A tibble: 256 x 6
##    .pred_class...17 .pred_class...18 .pred_class...19 .pred_class...20
##    <fct>            <fct>            <fct>            <fct>
##  1 rock             rap              rnb              edm
##  2 latin            rap              edm              pop
##  3 pop              rap              rnb              edm
##  4 rnb              edm              rock             rap
##  5 edm              rap              latin            pop
##  6 edm              latin            rap              rnb
##  7 rap              pop              rock             edm
##  8 pop              edm              rnb              rock
##  9 rap              edm              rnb              latin
```

Will sending the observations our committee is in most disagreement about to the "oracle" prove to increase test accuracy better than random observations? See full code in slides Rmd files.



Other Active Learning Metrics

- Expected Model Change
- Expected Error Reduction
- Variance Reduction
- And more...

Imbalanced Classes

Typical examples of Imbalanced Classes scenarios

- Rare diseases: this dataset contains genetic data for 1,000 HIV patients, 206 out of 1,000 patients improved after 16 weeks of therapy
- Conversion/Sell/CTR rates: this dataset contains 10 days of Click-Through-Rate data for Avazu mobile ads, ~6.8M clicked out of ~40.4M
- Fraud detection: this dataset contains credit card transactions for a major European CC, 492 frauds out of 284,807 transactions

What's so difficult about imbalanced classes?

```
okcupid_pets <- as_tibble(read_rds("../data/okcupid3_imp_mice.rds'

idx <- read_rds("../data/okcupid3_idx.rda")
train_idx <- idx$train_idx
valid_idx <- idx$valid_idx
test_idx <- idx$test_idx

ok_train <- okcupid_pets[train_idx, ]
ok_valid <- okcupid_pets[valid_idx, ]

ok_train %>%
  count(pets) %>%
  mutate(pct = round(n / sum(n), 2))
```

```
## # A tibble: 2 x 3
##   pets      n   pct
##   <fct> <int> <dbl>
## 1 cats   1624  0.16
## 2 dogs   8376  0.84
```

What's a sure way to get 84% accuracy?

```
mod_glm <- glm(pets ~ ., data = ok_train, family = "binomial")
pred_glm <- 1 - predict(mod_glm, ok_valid, type = "response")

pred_glm_class <- ifelse(pred_glm > 0.5, "cats", "dogs")
true_class <- ok_valid$pets

table(true_class, pred_glm_class)
```

```
##           pred_glm_class
## true_class cats dogs
##       cats   66  407
##       dogs   62 2194
```

```
report_accuracy_and_auc(true_class, pred_glm)
```

```
## Setting direction: controls < cases
```

```
## AUC: 0.736
## ACC: 0.828
## Cats: Recall: 0.14
##       Precision: 0.516
## Dogs: Recall: 0.973
##       Precision: 0.844
```

Remedies for Imbalanced Classes

- Model level
  - Tuning parameters and Cutoff choice
  - Cost-aware training: Case weights and Prior probabilities
- Data level
  - Down sampling
  - Up sampling
  - Get more data and features from minority class, similar to Active Learning
- Change of Framework
  - Anomaly Detection
- One final word of wisdom

Tuning parameters and cutoff choice

A general good approach would be:

1. Choose a model to maximize AUC on one part of the training dataset (using resampling)
2. Choose a cutoff score on another part of the training dataset
3. Fitting the entire thing on all training set and checking on test set

But. You could incorporate your initial goal even into (1), making the cutoff another tuning parameter that would maximize:

- Recall(cats): If never missing a cat person (the minority class) is your job (while maintaining acceptable level of Precision(cats))
- Precision(cats): If you don't have room for error when you say a person is a cat person (while maintaining acceptable level of Precision(cats))
- Some other metric like F1-score

Let us choose a model by maximizing AUC then present our client with a few potential cutoffs.

We'll begin by splitting our training set into two:

```
ok_split <- initial_split(ok_train, prop = 0.7, strata = pets)

ok_train1 <- training(ok_split)
ok_train2 <- testing(ok_split)

dim(ok_train1)
```

```
## [1] 7001   38
```

```
dim(ok_train2)
```

```
## [1] 2999   38
```

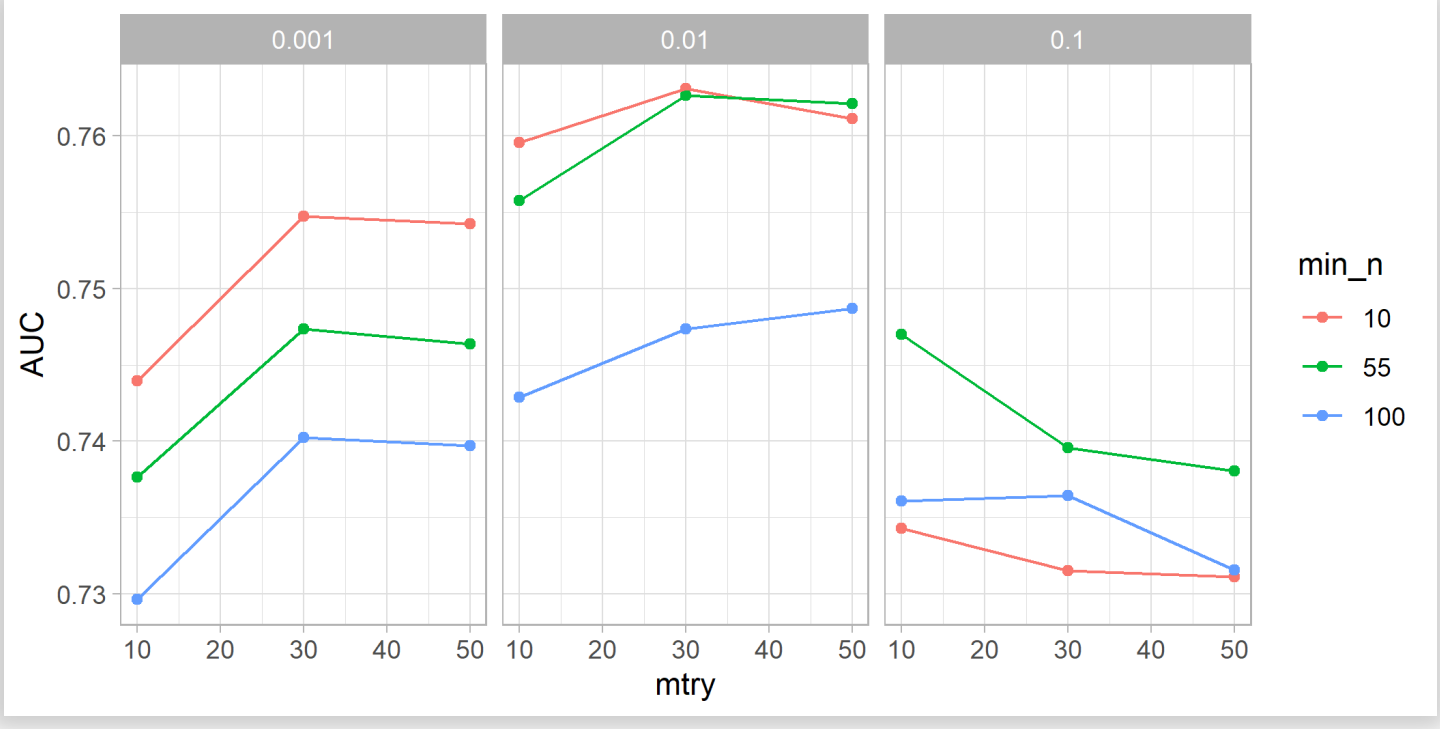
Use the first training set to choose a GBT model to maximize AUC with 5-fold CV:

```
mod_gbt_spec <- boost_tree(mode = "classification",
                           mtry = tune(),
                           min_n = tune(),
                           learn_rate = tune(),
                           trees = 1000) %>%
  set_engine("xgboost", eval_metric = "logloss")

gbt_grid <- grid_regular(mtry(range(10, 50)),
                         min_n(range(10, 100)),
                         learn_rate(range(-3, -1)),
                         levels = 3)
rec_gbt <- recipe(pets ~ ., data = ok_train1) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  prep(ok_train1)

cv_splits <- vfold_cv(juice(rec_gbt), v = 5, strata = pets)
```

```
tune_res <- tune_grid(object = mod_gbt_spec,
                      preprocessor = recipe(pets~., data = juice(r
                      resamples = cv_splits,
                      grid = gbt_grid,
                      control = control_grid(verbose = TRUE),
                      metrics = metric_set(roc_auc))
```

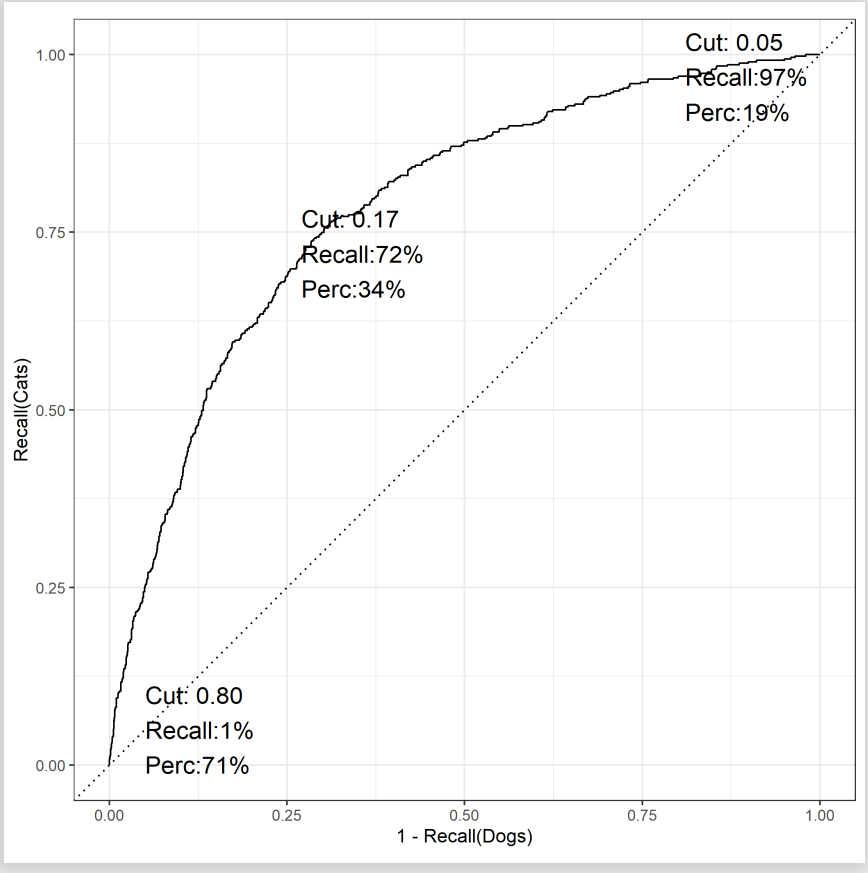


Fit the best model on all of `ok_train1` and get scores on `ok_train2`:

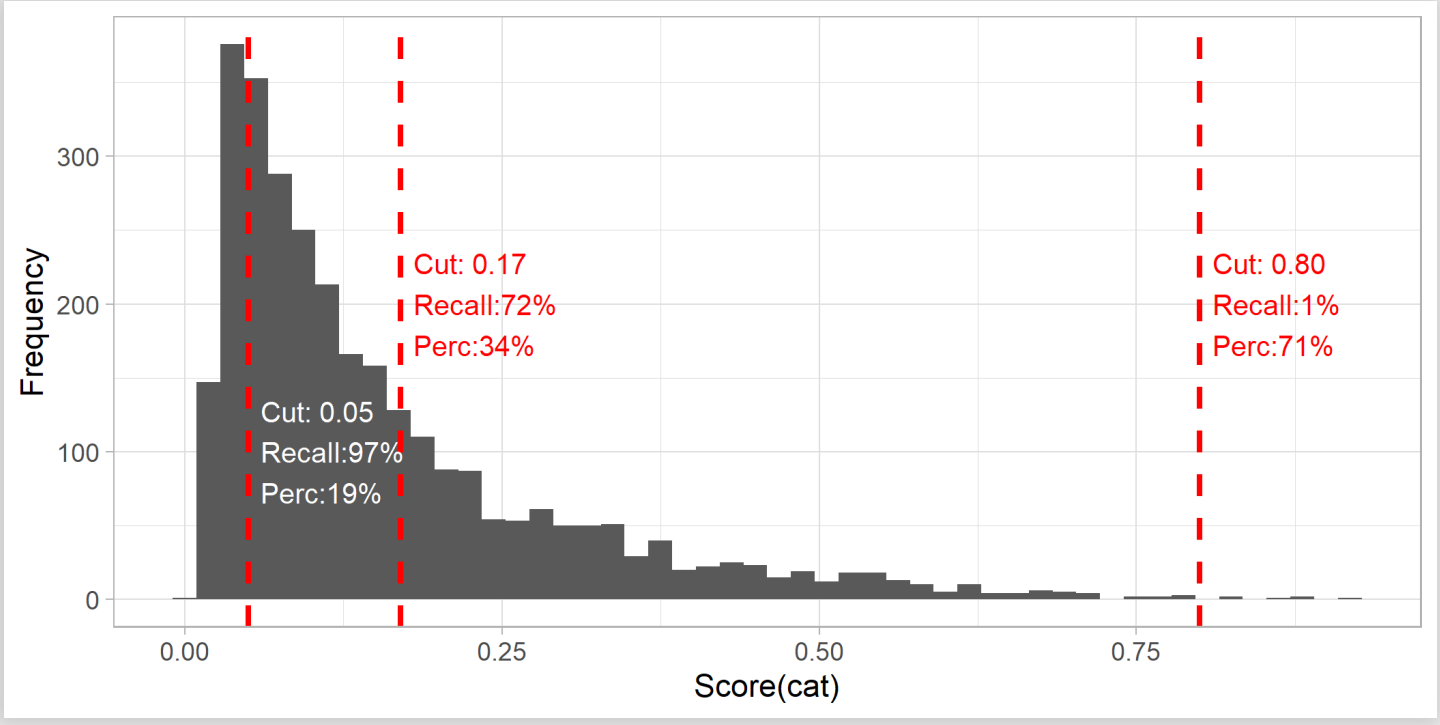
```
mod_gbt_spec <- mod_gbt_spec %>%
  update(mtry = 30, trees = 1000,
         min_n = 10, learn_rate = 0.01)
mod_gbt <- mod_gbt_spec %>%
  fit(pets ~ ., data = juice(rec_gbt))

pred_gbt <- mod_gbt %>%
  predict(new_data = bake(rec_gbt, ok_train2), type = "prob") %>%
  pull(.pred_cats)
```

You can use the ROC curve to understad the behavior of the cutoff:



Maybe better, draw a histogram of `cats` score and mark the cutoffs there:



Lastly, train on entire training set and evaluate on test set:

```
mod_gbt <- mod_gbt_spec %>%
  fit(pets ~ ., data = bake(rec_gbt, ok_train))
pred_gbt <- mod_gbt %>%
  predict(new_data = bake(rec_gbt, ok_valid), type = "prob") %>%
  pull(.pred_cats)
pred_gbt_class <- ifelse(pred_gbt > 0.17, "cats", "dogs")
true_class <- ok_valid$pets
table(true_class, pred_gbt_class)
```

```
##            pred_gbt_class
## true_class cats dogs
##       cats  309  164
##       dogs  647 1609
```

```
report_accuracy_and_auc(true_class, pred_gbt, cutoff = 0.17)
```

```
## AUC: 0.75
## ACC: 0.703
## Cats: Recall: 0.653
##       Precision: 0.323
## Dogs: Recall: 0.713
##       Precision: 0.908
```

Cost aware training: Case weights

For example, in `glm()` you can simply specify a `weights` param:

> when the elements of weights are positive integers *w_i*, each response *y_i* is the mean of *w_i* unit-weight observations

```
pets_weights <- rep(1, nrow(ok_train))
pets_weights[which(ok_train$pets == "cats")] <- 5

mod_glm <- glm(pets ~ ., data = ok_train,
               family = "binomial", weights = pets_weights)

pred_glm <- 1 - predict(mod_glm, ok_valid, type = "response")

pred_glm_class <- ifelse(pred_glm > 0.5, "cats", "dogs")
true_class <- ok_valid$pets
```

```
table(true_class, pred_glm_class)
```

```
##           pred_glm_class
## true_class cats dogs
##       cats  303  170
##       dogs  655 1601
```

```
report_accuracy_and_auc(true_class, pred_glm)
```

```
## AUC: 0.736
## ACC: 0.698
## Cats: Recall: 0.641
##       Precision: 0.316
## Dogs: Recall: 0.71
##       Precision: 0.904
```

But this is almost equivalent to up sampling.

A more intelligent use of class weights would be something like using the `class.weights` parameter in `e1071::svm()`

Cost aware training: Prior probabilities

Small Detour: Naive Bayes

You know Bayes' Theorem, right?

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

or

$$posterior = \frac{likelihood \cdot prior}{evidence}$$

So what would be the posterior probability of class C_k given that we've seen observation x_i?

$$P(C_k|x_i) = \frac{P(x_i|C_k)P(C_k)}{P(x_i)}$$

$$P(C_k|x_i) = \frac{P(x_i|C_k)P(C_k)}{P(x_i)}$$

In words: the likelihood of seeing an observation like x_i in all class C_k observations, times the prior of class C_k observations, divided by the evidence seeing an observation like x_i in general.

> What increases $P(C_k|x_i)$? What decreases it?

But if we have, say 100 predictors, each categorical with 2 levels - we'd have to pre-compute 2^{100} possible for each C_k!

Enter *Naive* Bayes:

Assume that all predictors X are mutually independent, conditional on the class C_k, and so:

$$P(x_i|C_k) = \prod_{j=1}^{p} P(x_{ij}|C_k)$$

And so:

$$P(C_k|x_i) = \frac{\prod P(x_{ij}|C_k)P(C_k)}{P(x_i)}$$

And we can further expand: $P(x_i) = \sum_k P(x_i|C_k)P(C_k)$

> How would you compute $P(x_{ij}|C_k)$ when x_{ij} is continuous?

```r
library(naivebayes)

mod_nb <- naive_bayes(pets ~ ., data = ok_train)

pred_nb <- predict(mod_nb, ok_valid, type = "prob")[, "cats"]

pred_nb_class <- ifelse(pred_nb > 0.5, "cats", "dogs")
table(true_class, pred_nb_class)
```

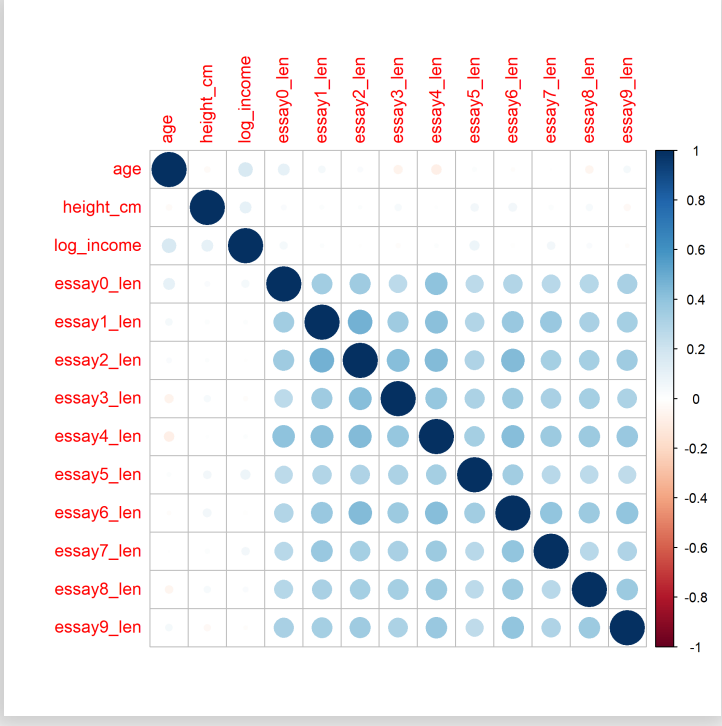
```
##           pred_nb_class
## true_class cats dogs
##       cats  180  293
##       dogs  294 1962
```

```r
report_accuracy_and_auc(true_class, pred_nb)
```

```
## AUC: 0.718
## ACC: 0.785
## Cats: Recall: 0.381
##       Precision: 0.38
## Dogs: Recall: 0.87
##       Precision: 0.87
```

BTW, *are* our features mutually independent?

```
ok_train %>%
  filter(pets == "cats") %>%
  select_if(is.numeric) %>%
  cor() %>%
  corrplot::corrplot()
```



In the context of imbalanced classes you could just give a 5 times more weight to the score of cats by specifying different prior probabilities $P(C_k)$:

```
mod_nb <- naive_bayes(pets ~ ., data = ok_train, prior = c(5, 1))

pred_nb <- predict(mod_nb, ok_valid, type = "prob")[, "cats"]

pred_nb_class <- ifelse(pred_nb > 0.5, "cats", "dogs")
table(true_class, pred_nb_class)
```

```
##           pred_nb_class
## true_class cats dogs
##       cats  398   75
##       dogs 1251 1005
```

```
report_accuracy_and_auc(true_class, pred_nb)
```

```
## AUC: 0.718
## ACC: 0.514
## Cats: Recall: 0.841
##       Precision: 0.241
## Dogs: Recall: 0.445
##       Precision: 0.931
```

Down Sampling

Yes, down sampling the majority class, usually to make it the same amount as the minority class (but you can tune this parameter as any other).

You'd be surprised.

```
rec_gbt <- recipe(pets ~ ., data = ok_train) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  themis::step_downsample(pets, under_ratio = 1) %>%
  prep(ok_train)
```

If you want to stay in the `tidymodels` framework you can download the `themis` package for extra recipes for dealing with unbalanced data.

⚠ Never down-sample the testing set! Look at the `skip` parameter.

```
juice(rec_gbt) %>% count(pets)
```

```
## # A tibble: 2 x 2
##   pets      n
##   <fct> <int>
## 1 cats   1624
## 2 dogs   1624
```

```
bake(rec_gbt, ok_valid) %>% count(pets)
```

```
## # A tibble: 2 x 2
##   pets      n
##   <fct> <int>
## 1 cats    473
## 2 dogs   2256
```

```
mod_gbt <- mod_gbt_spec %>%
  fit(pets ~ ., data = juice(rec_gbt))
pred_gbt <- mod_gbt %>%
  predict(new_data = bake(rec_gbt, ok_valid), type = "prob") %>%
  pull(.pred_cats)
pred_gbt_class <- ifelse(pred_gbt > 0.5, "cats", "dogs")

table(true_class, pred_gbt_class)
```

```
##           pred_gbt_class
## true_class cats dogs
##       cats  329  144
##       dogs  749 1507
```

```
report_accuracy_and_auc(true_class, pred_gbt)
```

```
## AUC: 0.734
## ACC: 0.673
## Cats: Recall: 0.696
##       Precision: 0.305
## Dogs: Recall: 0.668
##       Precision: 0.913
```

Up Sampling

The main disadvantage of down sampling is of course the loss of data.

Will replicating (minority class) data do any better?

```
rec_gbt <- recipe(pets ~ ., data = ok_train) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  themis::step_upsample(pets, over_ratio = 1) %>%
  prep(ok_train)

mod_gbt <- mod_gbt_spec %>%
  fit(pets ~ ., data = juice(rec_gbt))
pred_gbt <- mod_gbt %>%
  predict(new_data = bake(rec_gbt, ok_valid), type = "prob") %>%
  pull(.pred_cats)
pred_gbt_class <- ifelse(pred_gbt > 0.5, "cats", "dogs")
```

```
table(true_class, pred_gbt_class)
```

```
##           pred_gbt_class
## true_class cats dogs
##       cats  253  220
##       dogs  467 1789
```

```
report_accuracy_and_auc(true_class, pred_gbt)
```

```
## AUC: 0.74
## ACC: 0.748
## Cats: Recall: 0.535
##       Precision: 0.351
## Dogs: Recall: 0.793
##       Precision: 0.89
```

SMOTE

Chawla et. al. (2002) developed SMOTE (Synthetic Minority Over-sampling Technique) which is a up sampling technique.

The authors claim that a hybrid combination of SMOTE and regular down sampling works best, that's why SMOTE is sometimes referred to as a "hybrid" sampling algo itself.

But the up sampling does not simply replicates the minority class...

It synthesizes them!

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples

(∗ *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* ∗)
if $N < 100$
  then Randomize the T minority class samples
    $T = (N/100) * T$
    $N = 100$
endif
$N = (int)(N/100)$ (∗ *The amount of SMOTE is assumed to be in integral multiples of 100.* ∗)
k = Number of nearest neighbors
$numattrs$ = Number of attributes
$Sample[\,][\,]$: array for original minority class samples
$newindex$: keeps a count of number of synthetic samples generated, initialized to 0
$Synthetic[\,][\,]$: array for synthetic samples
(∗ *Compute k nearest neighbors for each minority class sample only.* ∗)
for $i \leftarrow 1$ **to** T
    Compute k nearest neighbors for i, and save the indices in the $nnarray$
    Populate(N, i, $nnarray$)
endfor

$Populate(N,\ i,\ nnarray)$ (∗ *Function to generate the synthetic samples.* ∗)
while $N \neq 0$
    Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
    for $attr \leftarrow 1$ **to** $numattrs$
      Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
      Compute: gap = random number between 0 and 1
      $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
    endfor
    $newindex{+}{+}$
    $N = N - 1$

```r
n <- 100
x1 <- rnorm(n, 0, 1); x2 <- rnorm(n, 0, 1)
t <- 2 - 4 * x1 + 3 * x2
y <- rbinom(n, 1, 1 / (1 + exp(-t)))

df <- tibble(x1 = x1, x2 = x2, y = factor(y))
df %>% count(y)
```

```
## # A tibble: 2 x 2
##   y         n
##   <fct> <int>
## 1 0        30
## 2 1        70
```

```r
df_smoted <- recipe(y ~ ., data = df) %>%
  themis::step_smote(y, over_ratio = 1) %>%
  prep(df) %>%
  juice()
```

```
df_smoted %>% count(y)
```

```
## # A tibble: 2 x 2
##   y         n
##   <fct> <int>
## 1 0        70
## 2 1        70
```

Let's do a hybrid of down sampling and SMOTE on our data:

```
rec_gbt <- recipe(pets ~ ., data = ok_train) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  themis::step_downsample(pets, under_ratio = 1.5) %>%
  themis::step_smote(pets, over_ratio = 1) %>%
  prep(ok_train)

juice(rec_gbt) %>% count(pets)
```

```
## # A tibble: 2 x 2
##   pets      n
##   <fct> <int>
## 1 cats   2436
## 2 dogs   2436
```

```
mod_gbt <- mod_gbt_spec %>%
  fit(pets ~ ., data = juice(rec_gbt))
pred_gbt <- mod_gbt %>%
  predict(new_data = bake(rec_gbt, ok_valid), type = "prob") %>%
  pull(.pred_cats)
pred_gbt_class <- ifelse(pred_gbt > 0.5, "cats", "dogs")
```

```
table(true_class, pred_gbt_class)
```

```
##           pred_gbt_class
## true_class cats dogs
##       cats  273  200
##       dogs  478 1778
```

```
report_accuracy_and_auc(true_class, pred_gbt)
```

```
## AUC: 0.753
## ACC: 0.752
## Cats: Recall: 0.577
##       Precision: 0.364
## Dogs: Recall: 0.788
##       Precision: 0.899
```

Other Up Sampling Methods

- ADASYN
- Borderline SMOTE
- ROSE
- Depends on data (e.g. with images it is common to perform image augmentation: flip, crop, rotate, blur it)

Change of Framework

- Anomaly Detection

One final word of wisdom

- It's Ok for a model to not know!
- The optimal choice would be for a classification model to output a *score*, rather than a class, and have the client's system interpret that score for different applications
- However, if a class output is required, consider outputting a $k+1$ class: "I don't know"
- In the case of classifying cats vs. dogs people - it makes sense!
- For a two-class problem, you would have not one cutoff on the score, but two: Below cutoff 1 classify as "Dogs", above "Cats" and in the middle: "I don't know"
- As long as you make a decision regarding at least X% of the data

Let's see this on the `ok_valid` test set using our last SMOTEd model (but notice to tune these cutoffs you would need an extra set of untouched data!):

```
upper <- 0.65
lower <- 0.35
pred_gbt_class <- ifelse(pred_gbt > upper, "cats",
                         ifelse(pred_gbt < lower, "dogs", NA))
table(true_class, pred_gbt_class)
```

```
##            pred_gbt_class
## true_class cats dogs
##       cats  143  101
##       dogs  195 1366
```

```r
report_accuracy_and_auc2 <- function(obs, pred, lower = 0.35, uppe
  pred_class <- ifelse(pred > upper, "cats",
                       ifelse(pred < lower, "dogs", NA))
  cm <- table(true_class, pred_class)
  recall_cats <- cm[1, 1] / sum(cm[1,])
  recall_dogs <- cm[2, 2] / sum(cm[2,])
  prec_cats <- cm[1, 1] / sum(cm[,1])
  prec_dogs <- cm[2, 2] / sum(cm[,2])
  acc <- sum(diag(cm)) / sum(cm)
  pred_pct <- sum(cm) / length(obs)
  glue::glue("Predicted: {format(pred_pct, digits = 3)}
  ACC: {format(acc, digits = 3)}
  Cats: Recall: {format(recall_cats, digits = 3)}
        Precision: {format(prec_cats, digits = 3)}
  Dogs: Recall: {format(recall_dogs, digits = 3)}
        Precision: {format(prec_dogs, digits = 3)}")
}

report_accuracy_and_auc2(true_class, pred_gbt)
```

```
## Predicted: 0.661
## ACC: 0.836
## Cats: Recall: 0.586
##       Precision: 0.423
## Dogs: Recall: 0.875
##       Precision: 0.931
```