

# APPLICATIONS



OF DATA SCIENCE

# Topics in Classification

## Applications of Data Science - Class 13

Giora Simchoni

[gsimchoni@gmail.com](mailto:gsimchoni@gmail.com) and add #dsapps in subject

Stat. and OR Department, TAU

2022-12-26

APPLICATIONS



OF DATA SCIENCE

# Topics in Classification

APPLICATIONS



OF DATA SCIENCE

# Life isn't perfect

Let's tackle just a few issues:

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

# Active Learning

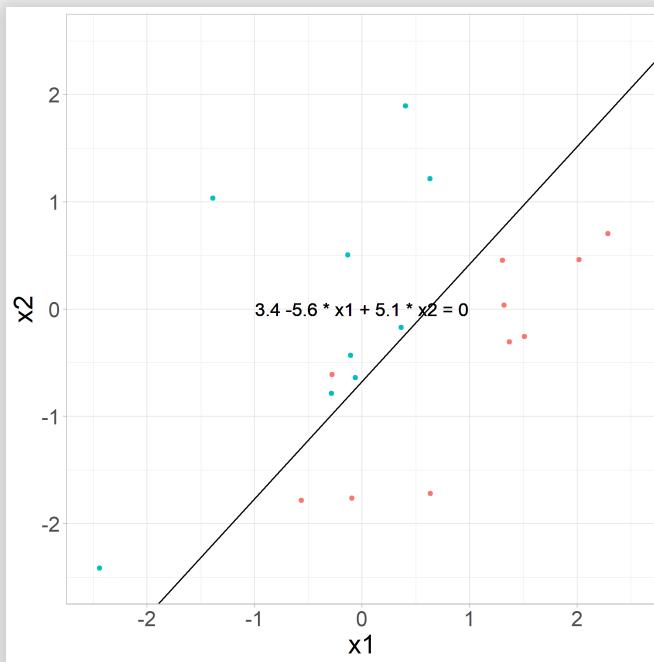
APPLICATIONS



OF DATA SCIENCE

# 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?

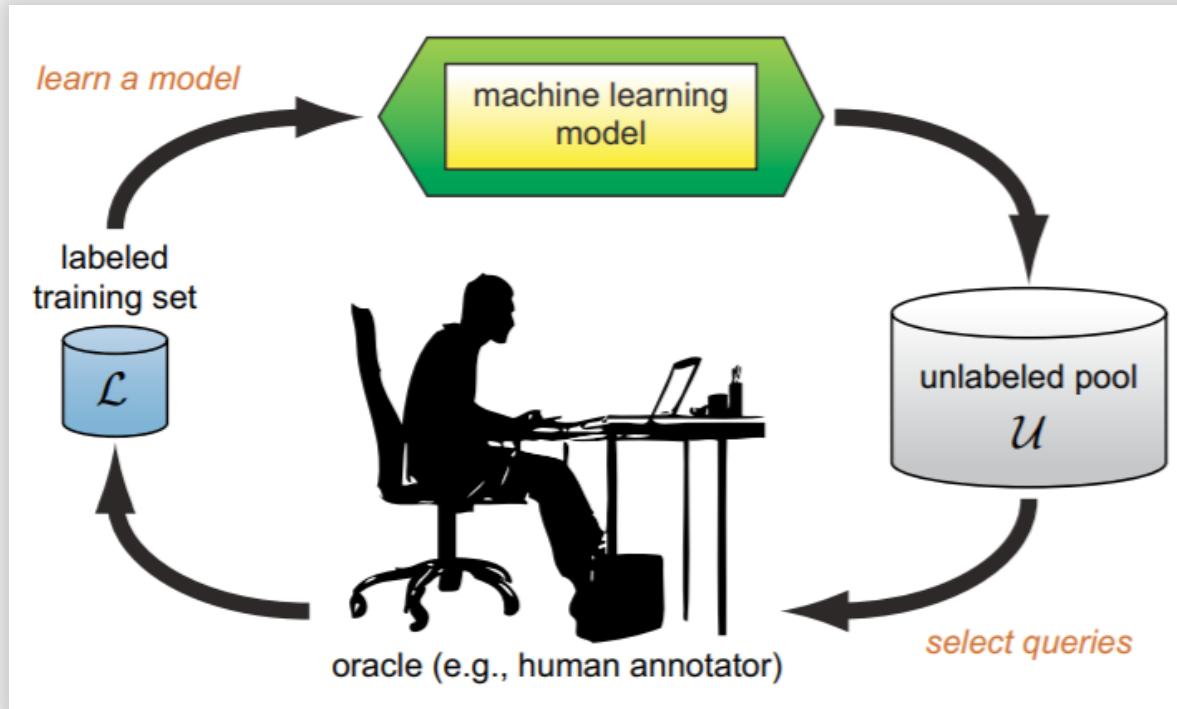
The key idea behind *active learning* is that a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns. An active learner may pose *queries*, usually in the form of unlabeled data instances to be labeled by an *oracle* (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!

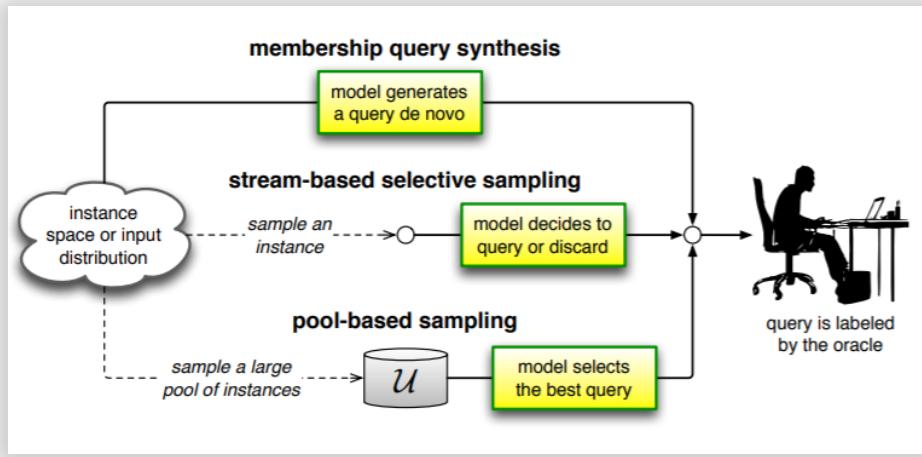
(No one, ever)

# Where this is going



# Active Learning Scenarios

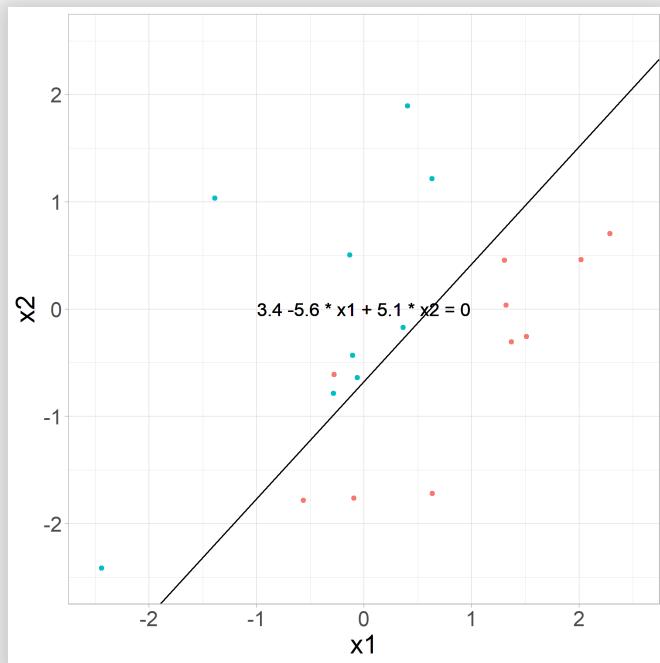
- 1. Membership Query Synthesis:** You get to choose which (maybe theoretical) points you'd want  $y$  labelled for.
- 2. Stream-Based Selective Sampling:** You get 1 point at a time and decide which ones you'd like to query and which to discard.
- 3. Pool-Based Sampling:** You have a large collection 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  $i$ th highest score (probability), for observations  $x$  under some model  $\theta$ .

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/rfordatascience/tidytuesday/master/data/2020/2020-03-10/spotify_songs.csv')  
  
spotify_songs %>% count(playlist_genre)  
  
## # A tibble: 6 × 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!

```
set.seed(1)
sptfy_tr_small <- sptfy_tr %>%
  group_by(playlist_genre) %>%
  slice_sample(n = 20) %>%
  ungroup()

sptfy_tr_small %>% count(playlist_genre)

## # A tibble: 6 × 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: 18,640 × 6
##   .pred_edm .pred_latin .pred_pop .pred_rap .pred_rnb .pred_rock
##   <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 0.851     0.0785    0.0531    0.0105    0.00253   0.00413
## 2 0.0673    0.0383    0.123     0.00895   0.0165    0.746
## 3 0.00363   0.0233    0.00457   0.962     0.00426   0.00192
## 4 0.00710   0.612     0.321     0.0322    0.0157    0.0118
## 5 0.00596   0.000497  0.0213    0.00191   0.000541  0.970
## 6 0.456     0.0792    0.179     0.261     0.00740   0.0175
## 7 0.933     0.0141    0.0363    0.00172   0.00598   0.00926
## 8 0.0807    0.464     0.246     0.0934    0.106     0.00940
## 9 0.0225    0.00112   0.933     0.00173   0.0301    0.0117
## 10 0.129    0.367     0.379     0.0908   0.00663   0.0277
## # ... with 18,630 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 × 3
##   .metric   .estimator .estimate
##   <chr>     <chr>        <dbl>
## 1 accuracy  multiclass  0.380
```

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 × 3
##   playlist_genre recall precision
##   <fct>        <dbl>     <dbl>
## 1 edm          0.556     0.550
## 2 latin         0.262     0.269
## 3 pop           0.236     0.231
## 4 rap           0.438     0.453
## 5 rnb           0.278     0.281
## 6 rock          0.459     0.438
```

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)  
  )  
}
```

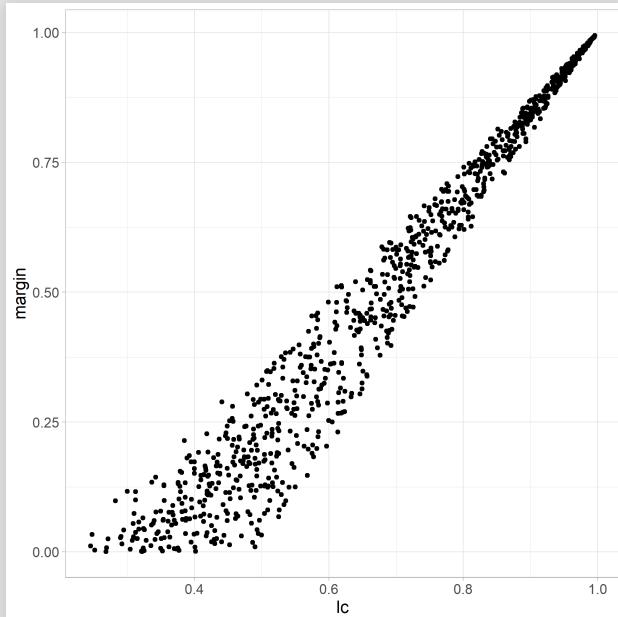
```
mod_unc <- mod_pred %>% pmap_dfr(uncertainty)

mod_unc
```

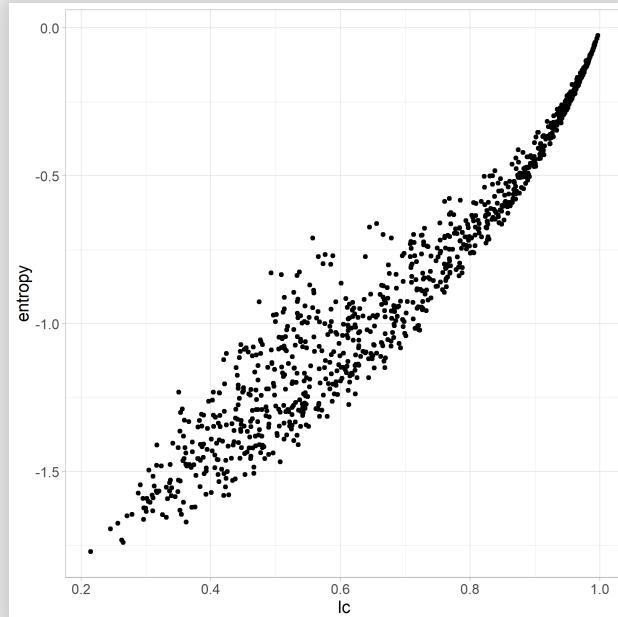
```
## # A tibble: 18,640 × 3
##       lc margin entropy
##   <dbl>  <dbl>    <dbl>
## 1 0.851  0.773   -0.578
## 2 0.746  0.623   -0.893
## 3 0.962  0.939   -0.205
## 4 0.612  0.291   -0.929
## 5 0.970  0.948   -0.162
## 6 0.456  0.195   -1.32
## 7 0.933  0.896   -0.330
## 8 0.464  0.218   -1.41
## 9 0.933  0.903   -0.326
## 10 0.379  0.0123 -1.35
## # ... with 18,630 more rows
```

Obviously these are correlated:

```
mod_unc %>% slice_sample(n = 1000) +  
  ggplot(aes(lc, margin)) +  
  geom_point() +  
  theme_light() +  
  theme(text =  
    element_text(size = 10))
```



```
mod_unc %>% slice_sample(n = 1000) +  
  ggplot(aes(lc, entropy)) +  
  geom_point() +  
  theme_light() +  
  theme(text =  
    element_text(size = 10))
```



## 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 × 6
##   track_name          track_artist playlist_genre   lc   margin
##   <chr>                <chr>           <fct>        <dbl>    <dbl>
## 1 Good Thing          Tritonal       edm            0.208  5.76e-3
## 2 Я смотрю на небо    Basta         rap            0.211  1.55e-2
## 3 Eenie Meenie        Justin Bieb... latin          0.215  1.40e-2
## 4 Can't Stop Me (feat. Sherm... Afrojack      edm            0.216  2.10e-2
## 5 Revival              Gregory Por... pop            0.222  1.04e-2
## 6 Take It To Reality  Alison Wond... pop            0.223  2.58e-2
## 7 Drops                Solarrio       rnb            0.224  1.52e-2
## 8 Fria Como El Viento Luis Miguel   latin          0.224  7.62e-5
## 9 Tu No Eres Para Mi  Fanny Lu     latin          0.224  2.27e-2
## 10 Symphony (feat. Zara Larss... Clean Bandit  edm            0.224  1.56e-4
```

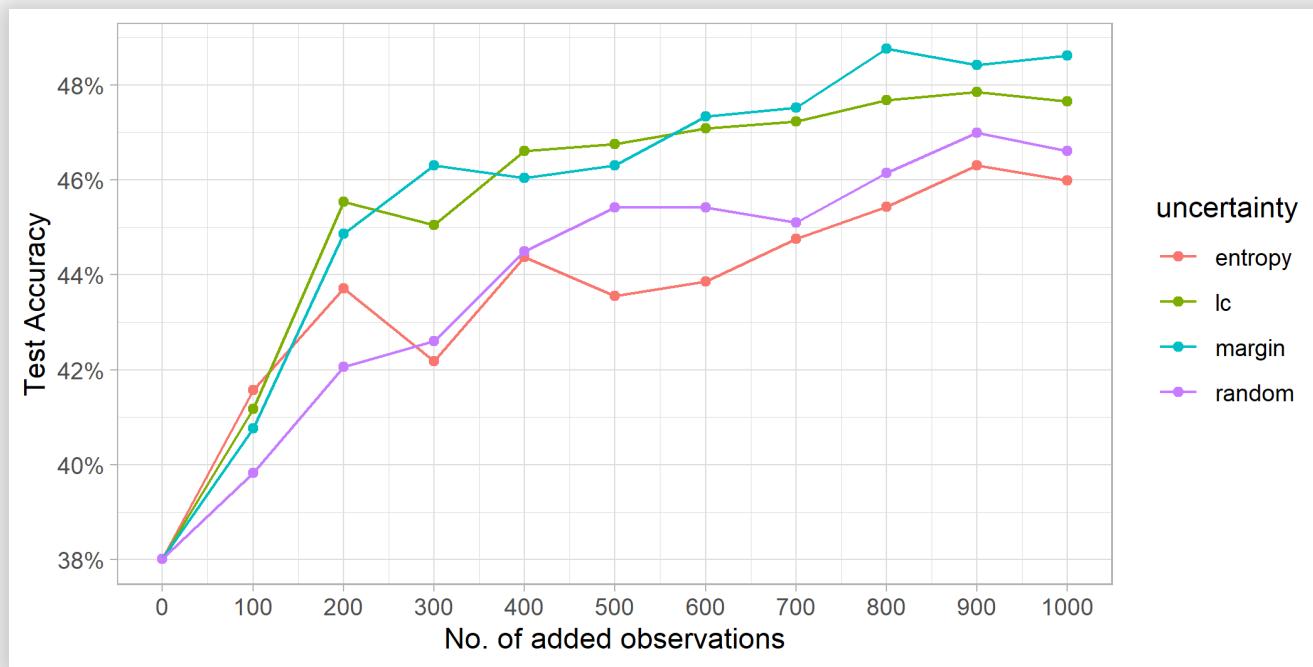
```
sptfy_tr_large_with_unc %>%
  slice_min(margin, n = 10) %>%
  arrange(margin, track_name)
```

## # A tibble: 10 × 6	## track_name	## <chr>	track_artist	playlist_genre	lc	margin
	1 Back in My Life		Alice DJ	pop	0.438	1.99e-5
	2 Fria Como El Viento		Luis Miguel	latin	0.224	7.62e-5
	3 Into The Night		Benny Mardo...	rock	0.226	1.01e-4
	4 El Taxi		Pitbull	latin	0.332	1.01e-4
	5 Symphony (feat. Zara Larss...		Clean Bandit	edm	0.224	1.56e-4
	6 I Can't Get No		JETFIRE	edm	0.407	1.63e-4
	7 Two Fine People		Yusuf / Cat...	rock	0.228	1.95e-4
	8 Y Si Te Digo - Version Mer...		Fanny Lu	latin	0.306	2.19e-4
	9 Under My Thumb		The Rolling...	rock	0.356	2.40e-4
	10 Vente Pa' Ca		Ricky Martin	latin	0.322	2.49e-4

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

```
## # A tibble: 10 × 6
##   track_name          track_artist playlist_genre   lc  margin
##   <chr>                <chr>           <fct>            <dbl> <dbl>
## 1 Я смотрю на небо      Basta          rap             0.211 1.55e-2
## 2 Eenie Meenie         Justin Bieber   latin            0.215 1.40e-2
## 3 Take It To Reality  Alison Wonderland pop              0.223 2.58e-2
## 4 Good Thing           Tritonal       edm              0.208 5.76e-3
## 5 Tu No Eres Para Mi  Fanny Lu        latin            0.224 2.27e-2
## 6 Into The Night        Benny Mardones rock             0.226 1.01e-4
## 7 Cold Ain't For Me   Oceans          pop              0.250 4.65e-2
## 8 Come Back             Green Tea       rnb              0.260 6.29e-2
## 9 Fortunate Son        Creedence Clearw... rock             0.235 3.61e-2
## 10 Never Getting Over You Lost Stars    pop              0.257 5.94e-2
```

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 Committee (QBC)

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

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

Which observations the committee 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: 18,640 × 6  
##   .pred_class...1 .pred_class...2 .pred_class...3 .pred_class...4  
##   <fct>          <fct>          <fct>          <fct>  
## 1 rap            pop             rnb            pop  
## 2 rnb            edm             edm            pop  
## 3 rock           rap             edm             rap  
## 4 rock           edm             rnb            rnb  
## 5 rock           latin           pop             rnb  
## 6 edm            rnb             rap             pop  
## 7 pop            rnb             pop             edm  
## 8 latin           rnb             edm             rap  
## 9 edm            rap             edm             pop  
## 10 latin          edm             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"))

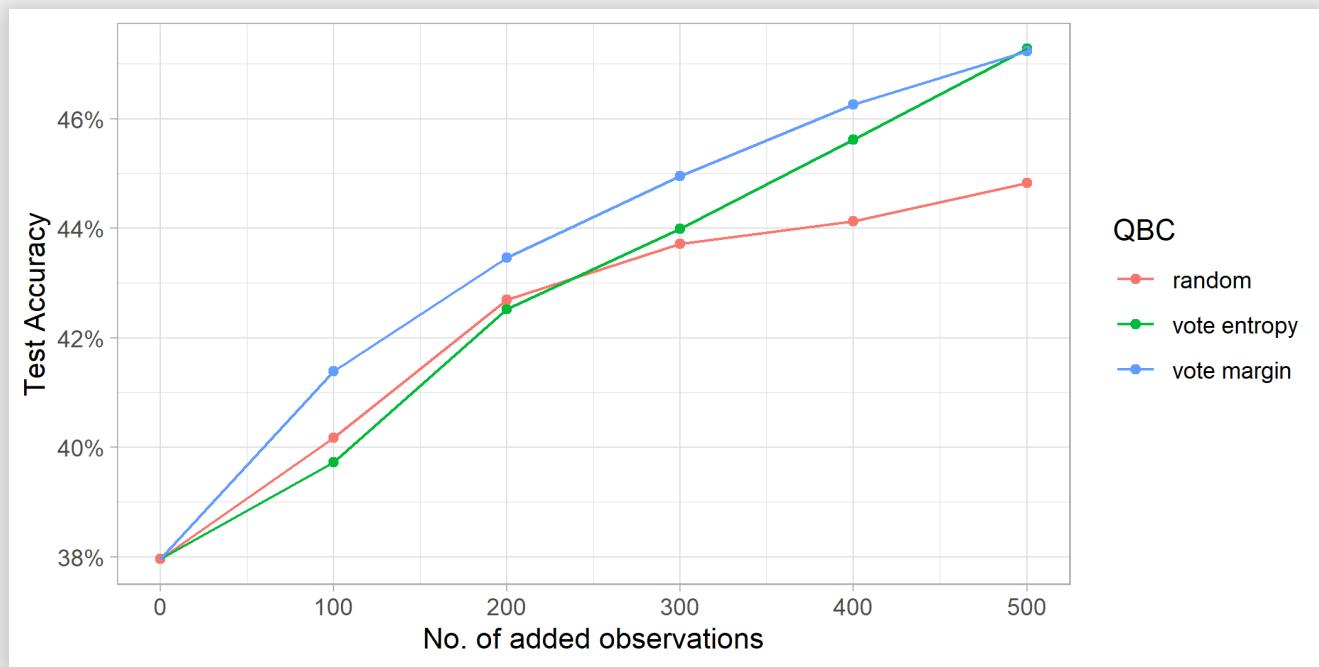
```

```

## # A tibble: 278 × 6
##   .pred_class...17 .pred_class...18 .pred_class...19 .pred_class...20
##   <fct>          <fct>          <fct>          <fct>
## 1 edm            pop             rap              rnb
## 2 rock           rap             edm              rnb
## 3 rock           pop             rap              edm
## 4 latin          rock            pop              edm
## 5 edm            pop             rap              latin
## 6 edm            pop             latin            rnb
## 7 edm            latin            pop              rap
## 8 rock           rnb             edm              latin
## 9 rnb            rock            rap              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

APPLICATIONS



OF DATA SCIENCE

# 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 × 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] 6999    38  
  
dim(ok_train2)  
  
## [1] 3001    38
```

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

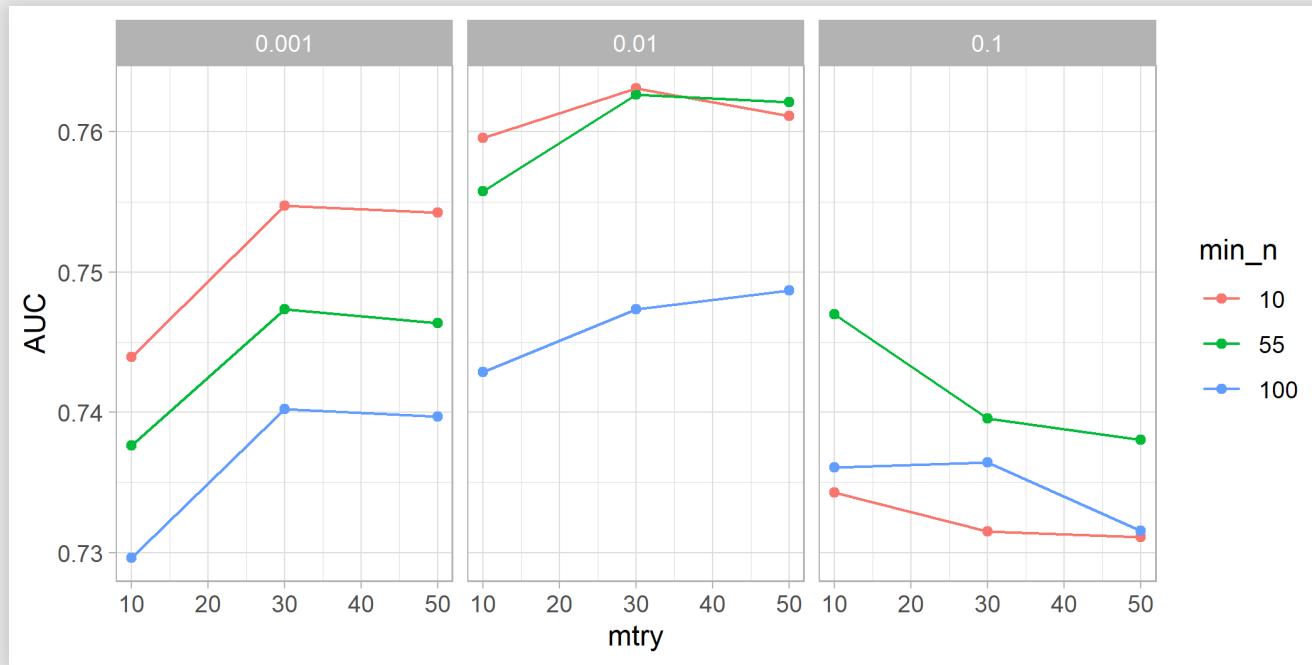
```
mod_gbt_spec <- boost_tree(mode = "classification",
                           mtry = tune(),
                           min_n = tune(),
                           learn_rate = tune(),
                           trees = 1000) %>%
  set_engine("xgboost", eval_metric = "logloss")

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))
```

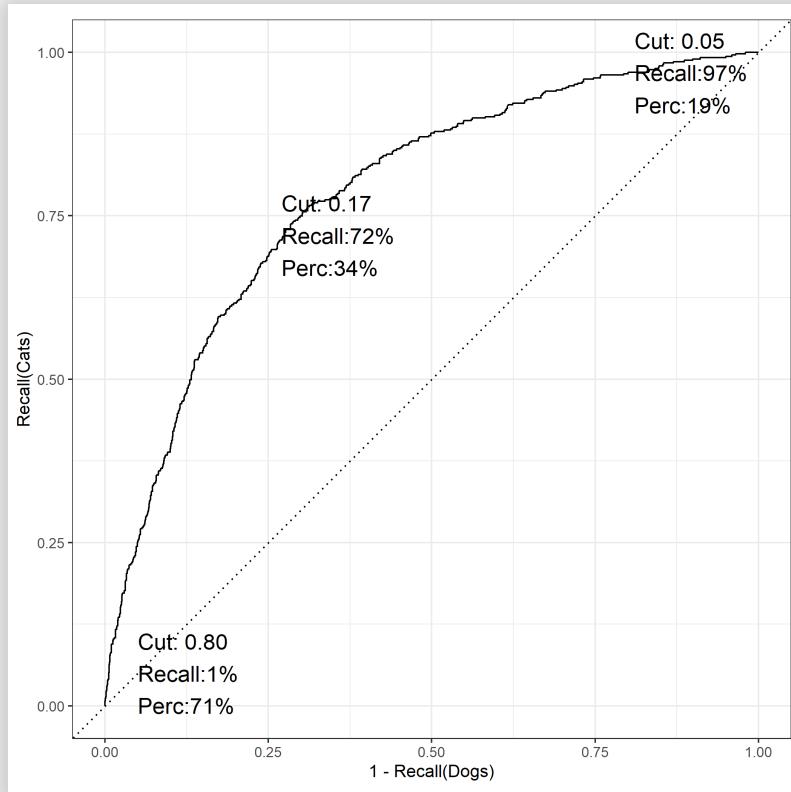


Fit the best model on all of `ok_train1` and get scores on `ok_train2`:

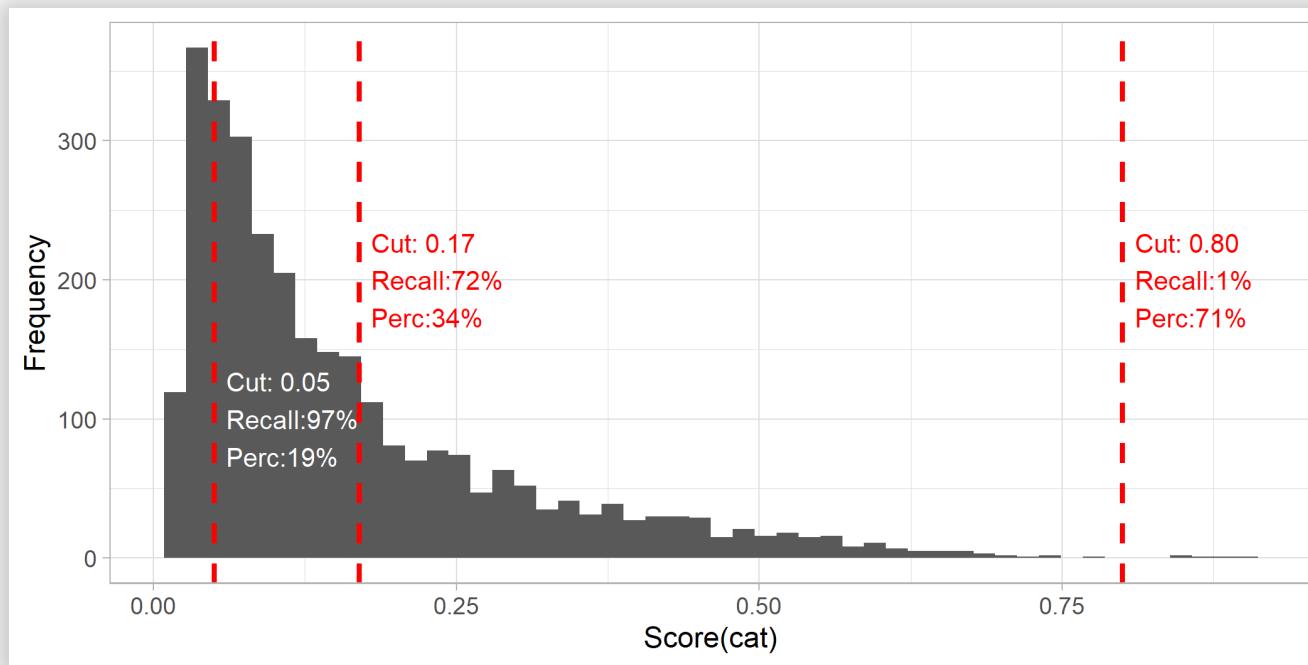
```
mod_gbt_spec <- mod_gbt_spec %>%
  update(mtry = 30, trees = 1000,
         min_n = 10, learn_rate = 0.01)
mod_gbt <- mod_gbt_spec %>%
  fit(pets ~ ., data = juice(rec_gbt))

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

You can use the ROC curve to understand 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    307   166
##       dogs    649 1607
```

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

```
## AUC: 0.746
## ACC: 0.701
## Cats: Recall: 0.649
##          Precision: 0.321
## Dogs: Recall: 0.712
##          Precision: 0.906
```

# 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

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

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  $2^{100}$  possibilities 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?

```
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)
```

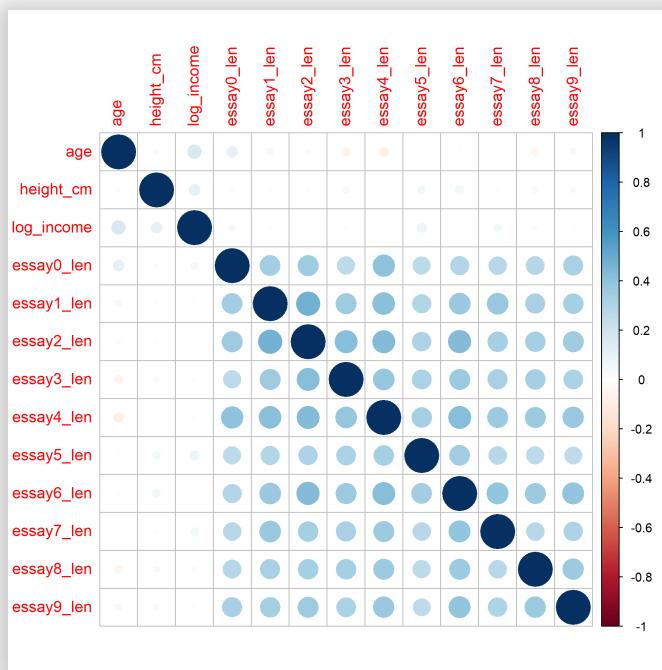
```
##           pred_nb_class
## true_class  cats  dogs
##       cats    180   293
##       dogs    294 1962
```

```
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 × 2
##   pets     n
##   <fct> <int>
## 1 cats    1624
## 2 dogs    1624
```

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

```
## # A tibble: 2 × 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    323   150
##       dogs    733 1523
```

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

```
## AUC: 0.734
## ACC: 0.676
## Cats: Recall: 0.683
##         Precision: 0.306
## Dogs: Recall: 0.675
##        Precision: 0.91
```

# 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    262   211
##      dogs    497 1759
```

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

```
## AUC: 0.739
## ACC: 0.741
## Cats: Recall: 0.554
##          Precision: 0.345
## Dogs: Recall: 0.78
##          Precision: 0.893
```

# 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
1. (* If N is less than 100%, randomize the minority class samples as only a random
percent of them will be SMOTEd. *)
2. if N < 100
3.   then Randomize the T minority class samples
4.   T = (N/100) * T
5.   N = 100
6. endif
7. N = (int)(N/100) (* The amount of SMOTE is assumed to be in integral multiples of
100. *)
8. k = Number of nearest neighbors
9. numattrs = Number of attributes
10. Sample[ ][ ]: array for original minority class samples
11. newindex: keeps a count of number of synthetic samples generated, initialized to 0
12. Synthetic[ ][ ]: array for synthetic samples
(* Compute k nearest neighbors for each minority class sample only. *)
13. for i ← 1 to T
14.   Compute k nearest neighbors for i, and save the indices in the nnarray
15.   Populate(N, i, nnarray)
16. endfor

Populate(N, i, nnarray) (* Function to generate the synthetic samples. *)
17. while N ≠ 0
18.   Choose a random number between 1 and k, call it nn. This step chooses one of
the k nearest neighbors of i.
19.   for attr ← 1 to numattrs
20.     Compute: dif = Sample[nnarray[nn]][attr] - Sample[i][attr]
21.     Compute: gap = random number between 0 and 1
22.     Synthetic[newindex][attr] = Sample[i][attr] + gap * dif
23.   endfor
24.   newindex++
25.   N = N - 1
```

```
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)))

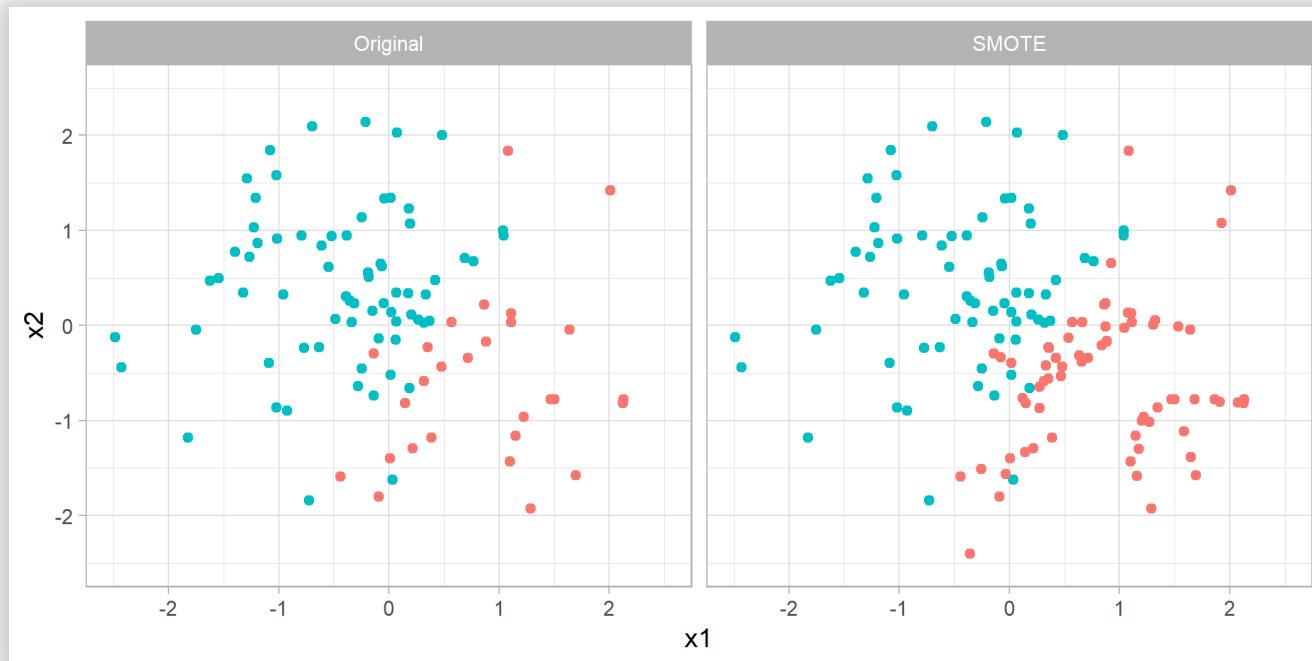
df <- tibble(x1 = x1, x2 = x2, y = factor(y))
df %>% count(y)
```

```
## # A tibble: 2 × 2
##   y     n
##   <fct> <int>
## 1 0      29
## 2 1      71
```

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

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

```
## # A tibble: 2 × 2
##   y     n
##   <fct> <int>
## 1 0      71
## 2 1      71
```



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 × 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    282   191
##      dogs    500 1756
```

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

```
## AUC: 0.744
## ACC: 0.747
## Cats: Recall: 0.596
##          Precision: 0.361
## Dogs: Recall: 0.778
##          Precision: 0.902
```

# 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    156   103
##      dogs    239 1324
```

```

report_accuracy_and_auc2 <- function(obs, pred, lower = 0.35, upper = 0.65) {
  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)}\n"
             "ACC: {format(acc, digits = 3)}\n"
             "Cats: Recall: {format(recall_cats, digits = 3)}\n"
             "      Precision: {format(prec_cats, digits = 3)}\n"
             "Dogs: Recall: {format(recall_dogs, digits = 3)}\n"
             "      Precision: {format(prec_dogs, digits = 3)}")
}
report_accuracy_and_auc2(true_class, pred_gbt)

```

```

## Predicted: 0.668
## ACC: 0.812
## Cats: Recall: 0.602
##          Precision: 0.395
## Dogs: Recall: 0.847
##          Precision: 0.928

```