11 Subsampling For Class Imbalances

Contents

In classification problems, a disparity in the frequencies of the observed classes can have a significant negative impact on model fitting. One technique for resolving such a class imbalance is to subsample the training data in a manner that mitigates the issues. Examples of sampling methods for this purpose are:

- *down-sampling*: randomly subset all the classes in the training set so that their class frequencies match the least prevalent class. For example, suppose that 80% of the training set samples are the first class and the remaining 20% are in the second class. Down-sampling would randomly sample the first class to be the same size as the second class (so that only 40% of the total training set is used to fit the model). **caret** contains a function (`downSample`) to do this.
- *up-sampling*: randomly sample (with replacement) the minority class to be the same size as the majority class. **caret** contains a function (`upSample`) to do this.
- *hybrid methods*: techniques such as SMOTE and ROSE down-sample the majority class and synthesize new data points in the minority class. There are two packages (**DMwR** and **ROSE**) that implement these procedures.

Note that this type of sampling is different from splitting the data into a training and test set. You would never want to artificially balance the test set; its class frequencies should be in-line with what one would see "in the wild". Also, the above procedures are independent of resampling methods such as cross-validation and the bootstrap.

In practice, one could take the training set and, before model fitting, sample the data. There are two issues with this approach

- Firstly, during model tuning the holdout samples generated during resampling are also glanced and may not reflect the class imbalance that future predictions would encounter. This is likely to lead to overly optimistic estimates of performance.
- Secondly, the subsampling process will probably induce more model uncertainty. Would the model results differ under a different subsample? As above, the resampling statistics are more likely to make the model appear more effective than it actually is.

The alternative is to include the subsampling inside of the usual resampling procedure. This is also advocated for pre-process and featur selection steps too. The two disadvantages are that it might increase computational times and that it might also complicate the analysis in other ways (see the section below about the pitfalls).

11.1 Subsampling Techniques

To illustrate these methods, let's simulate some data with a class imbalance using this method. We will simulate a training and test set where each contains 10000 samples and a minority class rate of about 5.9%:

```r
library(caret)

set.seed(2969)
imbal_train <- twoClassSim(10000, intercept = -20, linearVars = 20)
imbal_test  <- twoClassSim(10000, intercept = -20, linearVars = 20)
table(imbal_train$Class)
```

```
##
## Class1 Class2
##   9411    589
```

Let's create different versions of the training set prior to model tuning:

```r
set.seed(9560)
down_train <- downSample(x = imbal_train[, -ncol(imbal_train)],
                         y = imbal_train$Class)
table(down_train$Class)
```

```
##
## Class1 Class2
##    589    589
```

```
set.seed(9560)
up_train <- upSample(x = imbal_train[, -ncol(imbal_train)],
                     y = imbal_train$Class)
table(up_train$Class)
```

```
##
## Class1 Class2
##   9411   9411
```

```
library(DMwR)

set.seed(9560)
smote_train <- SMOTE(Class ~ ., data  = imbal_train)
table(smote_train$Class)
```

```
##
## Class1 Class2
##   2356   1767
```

```
library(ROSE)

set.seed(9560)
rose_train <- ROSE(Class ~ ., data  = imbal_train)$data
table(rose_train$Class)
```

```
##
## Class1 Class2
##   4939   5061
```

For these data, we’ll use a bagged classification and estimate the area under the ROC curve using five repeats of 10-fold CV.

```r
ctrl <- trainControl(method = "repeatedcv", repeats = 5,
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary)

set.seed(5627)
orig_fit <- train(Class ~ ., data = imbal_train,
                  method = "treebag",
                  nbagg = 50,
                  metric = "ROC",
                  trControl = ctrl)

set.seed(5627)
down_outside <- train(Class ~ ., data = down_train,
                      method = "treebag",
                      nbagg = 50,
                      metric = "ROC",
                      trControl = ctrl)

set.seed(5627)
up_outside <- train(Class ~ ., data = up_train,
                    method = "treebag",
                    nbagg = 50,
                    metric = "ROC",
                    trControl = ctrl)

set.seed(5627)
rose_outside <- train(Class ~ ., data = rose_train,
                      method = "treebag",
                      nbagg = 50,
                      metric = "ROC",
                      trControl = ctrl)


set.seed(5627)
smote_outside <- train(Class ~ ., data = smote_train,
                       method = "treebag",
                       nbagg = 50,
                       metric = "ROC",
                       trControl = ctrl)
```

We will collate the resampling results and create a wrapper to estimate the test set performance:

```
outside_models <- list(original = orig_fit,
                       down = down_outside,
                       up = up_outside,
                       SMOTE = smote_outside,
                       ROSE = rose_outside)

outside_resampling <- resamples(outside_models)

test_roc <- function(model, data) {
  library(pROC)
  roc_obj <- roc(data$Class,
                 predict(model, data, type = "prob")[, "Class1"],
                 levels = c("Class2", "Class1"))
  ci(roc_obj)
  }

outside_test <- lapply(outside_models, test_roc, data = imbal_test)
outside_test <- lapply(outside_test, as.vector)
outside_test <- do.call("rbind", outside_test)
colnames(outside_test) <- c("lower", "ROC", "upper")
outside_test <- as.data.frame(outside_test)

summary(outside_resampling, metric = "ROC")
```

```
##
## Call:
## summary.resamples(object = outside_resampling, metric = "ROC")
##
## Models: original, down, up, SMOTE, ROSE
## Number of resamples: 50
##
## ROC
##            Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## original 0.8935  0.9311 0.9407 0.9410  0.9564 0.9717    0
## down     0.8937  0.9205 0.9369 0.9347  0.9549 0.9684    0
## up       0.9995  1.0000 1.0000 0.9999  1.0000 1.0000    0
## SMOTE    0.9622  0.9762 0.9808 0.9796  0.9840 0.9925    0
## ROSE     0.8764  0.8908 0.8953 0.8956  0.8999 0.9166    0
```

```
outside_test
```

```
##              lower       ROC     upper
## original 0.9134895 0.9251843 0.9368791
## down     0.9238817 0.9315572 0.9392327
## up       0.9360997 0.9437082 0.9513167
## SMOTE    0.9390808 0.9457264 0.9523720
## ROSE     0.9409081 0.9474742 0.9540403
```

The training and test set estimates for the area under the ROC curve do not appear to correlate. Based on the resampling results, one would infer that up-sampling is nearly perfect and that ROSE does relatively poorly. The reason that up-sampling appears to perform so well is that the samples in the majority class are replicated and have a large potential to be in both the model building and hold-out sets. In essence, the hold-outs here are not truly independent samples.

In reality, all of the sampling methods do about the same (based on the test set). The statistics for the basic model fit with no sampling are fairly in-line with one another (0.941 via resampling and 0.925 for the test set).

11.2 Subsampling During Resampling

Recent versions of **caret** allow the user to specify subsampling when using `train` so that it is conducted inside of resampling. All four methods shown above can be accessed with the basic package using simple syntax. If you want to use your own technique, or want to change some of the parameters for `SMOTE` or `ROSE`, the last section below shows how to use custom subsampling.

The way to enable subsampling is to use yet another option in `trainControl` called `sampling`. The most basic syntax is to use a character string with the name of the sampling method, either `"down"`, `"up"`, `"smote"`, or `"rose"`. Note that you will need to have the **DMwR** and **ROSE** packages installed to use SMOTE and ROSE, respectively.

One complication is related to pre-processing. Should the subsampling occur before or after the pre-processing? For example, if you down-sample the data and using PCA for signal extraction, should the loadings be estimated from the entire training set? The estimate is potentially better since the entire training set is being used but the subsample may happen to capture a small potion of the PCA space. There isn't any obvious answer.

The default behavior is to subsample the data prior to pre-processing. This can be easily changed and an example is given below.

Now let's re-run our bagged tree models while sampling inside of cross-validation:

```
ctrl <- trainControl(method = "repeatedcv", repeats = 5,
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary,
                     ## new option here:
                     sampling = "down")

set.seed(5627)
down_inside <- train(Class ~ ., data = imbal_train,
                     method = "treebag",
                     nbagg = 50,
                     metric = "ROC",
                     trControl = ctrl)

## now just change that option
ctrl$sampling <- "up"

set.seed(5627)
up_inside <- train(Class ~ ., data = imbal_train,
                   method = "treebag",
                   nbagg = 50,
                   metric = "ROC",
                   trControl = ctrl)

ctrl$sampling <- "rose"

set.seed(5627)
rose_inside <- train(Class ~ ., data = imbal_train,
                     method = "treebag",
                     nbagg = 50,
                     metric = "ROC",
                     trControl = ctrl)

ctrl$sampling <- "smote"

set.seed(5627)
smote_inside <- train(Class ~ ., data = imbal_train,
                      method = "treebag",
                      nbagg = 50,
                      metric = "ROC",
                      trControl = ctrl)
```

Here are the resampling and test set results:

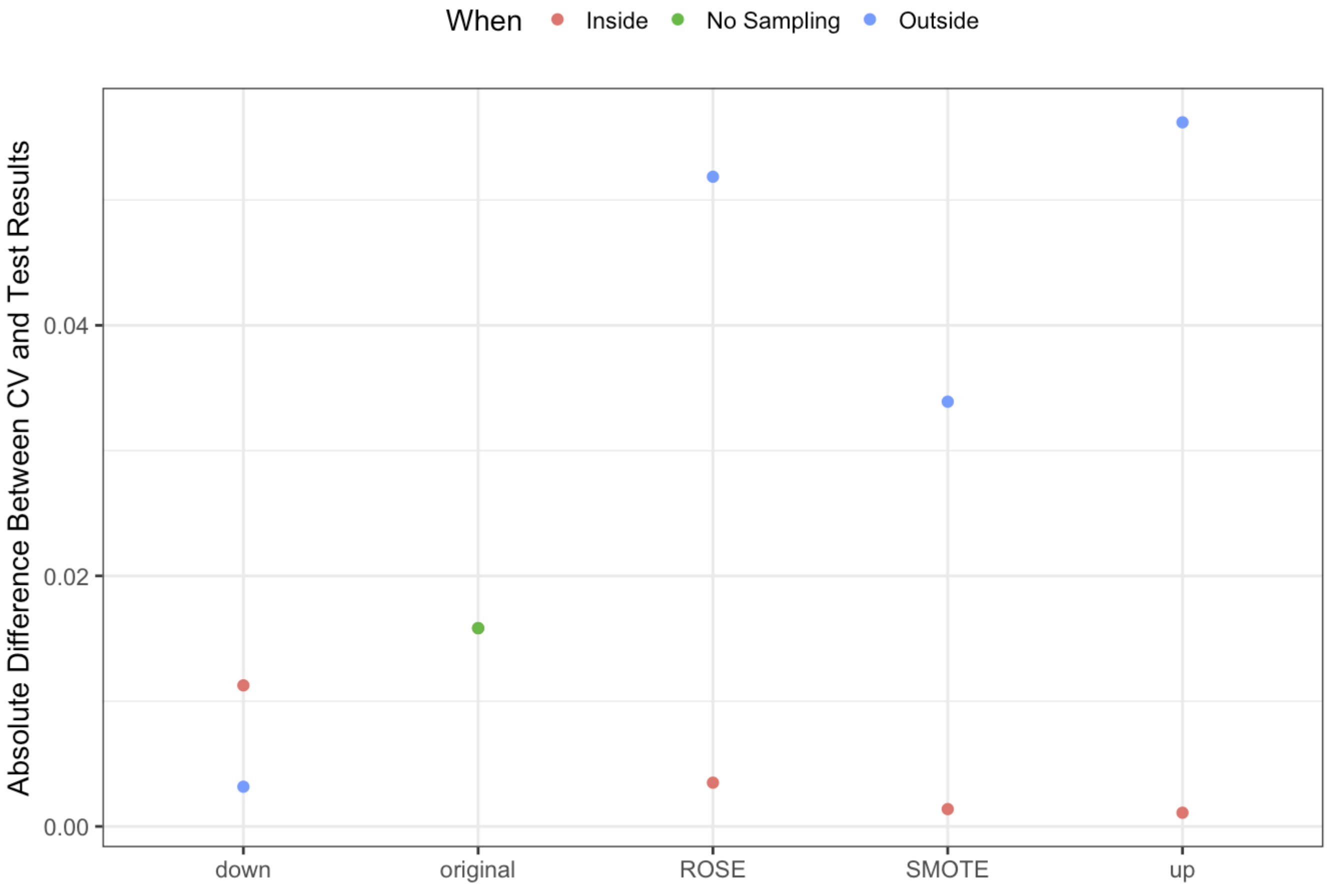
```
inside_models <- list(original = orig_fit,
                      down = down_inside,
                      up = up_inside,
                      SMOTE = smote_inside,
                      ROSE = rose_inside)

inside_resampling <- resamples(inside_models)

inside_test <- lapply(inside_models, test_roc, data = imbal_test)
inside_test <- lapply(inside_test, as.vector)
inside_test <- do.call("rbind", inside_test)
colnames(inside_test) <- c("lower", "ROC", "upper")
inside_test <- as.data.frame(inside_test)

summary(inside_resampling, metric = "ROC")
```

```
##
## Call:
## summary.resamples(object = inside_resampling, metric = "ROC")
##
## Models: original, down, up, SMOTE, ROSE
## Number of resamples: 50
##
## ROC
##            Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## original 0.8935  0.9311 0.9407 0.9410  0.9564 0.9717    0
## down     0.8902  0.9348 0.9445 0.9421  0.9507 0.9665    0
## up       0.8928  0.9211 0.9374 0.9352  0.9499 0.9637    0
## SMOTE    0.9299  0.9458 0.9528 0.9527  0.9609 0.9761    0
## ROSE     0.9189  0.9440 0.9529 0.9517  0.9597 0.9741    0
```

```
inside_test
```

```
##               lower       ROC     upper
## original 0.9134895 0.9251843 0.9368791
## down     0.9234409 0.9308037 0.9381665
## up       0.9241506 0.9340749 0.9439992
## SMOTE    0.9455682 0.9512705 0.9569727
## ROSE     0.9413039 0.9482492 0.9551946
```

The figure below shows the difference in the area under the ROC curve and the test set results for the approaches shown here. Repeating the subsampling procedures for every resample produces results that are more consistent with the test set.



11.3 Complications

The user should be aware that there are a few things that can happening when subsampling that can cause issues in their code. As previously mentioned, when sampling occurs in relation to pre-processing is one such issue. Others are:

- Sparsely represented categories in factor variables may turn into zero-variance predictors or may be completely sampled out of the model.

- The underlying functions that do the sampling (e.g. `SMOTE` , `downSample` , etc) operate in very different ways and this can affect your results. For example, `SMOTE` and `ROSE` will convert your predictor input argument into a data frame (even if you start with a matrix).
- Currently, sample weights are not supported with sub-sampling.
- If you use `tuneLength` to specify the search grid, understand that the data that is used to determine the grid has not been sampled. In most cases, this will not matter but if the grid creation process is affected by the sample size, you may end up using a sub-optimal tuning grid.
- For some models that require more samples than parameters, a reduction in the sample size may prevent you from being able to fit the model.

11.4 Using Custom Subsampling Techniques

Users have the ability to create their own type of subsampling procedure. To do this, alternative syntax is used with the `sampling` argument of the `trainControl` . Previously, we used a simple string as the value of this argument. Another way to specify the argument is to use a list with three (named) elements:

- The `name` value is a character string used when the `train` object is printed. It can be any string.
- The `func` element is a function that does the subsampling. It should have arguments called `x` and `y` that will contain the predictors and outcome data, respectively. The function should return a list with elements of the same name.
- The `first` element is a single logical value that indicates whether the subsampling should occur first relative to pre-process. A value of `FALSE` means that the subsampling function will receive the sampled versions of `x` and `y` .

For example, here is what the list version of the `sampling` argument looks like when simple down-sampling is used:

```
down_inside$control$sampling
```

```
## $name
## [1] "down"
##
## $func
## function (x, y)
## downSample(x, y, list = TRUE)
##
## $first
## [1] TRUE
```

As another example, suppose we want to use SMOTE but use 10 nearest neighbors instead of the default of 5. To do this, we can create a simple wrapper around the `SMOTE` function and call this instead:

```
smotest <- list(name = "SMOTE with more neighbors!",
                func = function (x, y) {
                  library(DMwR)
                  dat <- if (is.data.frame(x)) x else as.data.frame(x)
                  dat$.y <- y
                  dat <- SMOTE(.y ~ ., data = dat, k = 10)
                  list(x = dat[, !grepl(".y", colnames(dat), fixed = TRUE)],
                       y = dat$.y)
                  },
                first = TRUE)
```

The control object would then be:

```
ctrl <- trainControl(method = "repeatedcv", repeats = 5,
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary,
                     sampling = smotest)
```