42-naivebayes-modeling

Code

In this notebook, we use Naive Bayes to perform modeling. Our general approach will be to use hyperparameter tuning via cross-validation to identify the best performing hyperparameters, In another notebook, we will investigate the performance of the model.

Note that prior to running this notebook, 10 and 40 must already have been run.

0.0.1 Useful packages

Hide

```
pacman::p_load(glmnet, tictoc, vip, tidytext, doParallel, caret, naivebayes)
library(discrim)
```

Here, we define the specs for the feature engineering, the model, the generalized workflow, and the parameters that we’ll tune using parameters selected from a max entropy grid. For the Naive Bayes model, the general usemodels template code was not supported but using the naivebayes package model allowed for easy 1:1 translation

Hide

```
nb_recipe <-
  recipe(formula = particle_class ~ ., data = train_data) %>%
  update_role(id, img_id, starts_with('filter'), hash, new_role='no_model') %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors(), -all_nominal())

nb_spec <-
  naive_Bayes(smoothness = tune(), Laplace = tune() ) %>%
  set_mode("classification") %>%
  set_engine("naivebayes")

nb_workflow <-
  workflow() %>%
  add_recipe(nb_recipe) %>%
  add_model(nb_spec)

nb_parameters <- parameters(nb_spec)
nb_grid <- grid_max_entropy(nb_parameters, size=20)
```

1 Hyperparameter tuning (model selection) via cross-validation

Hide

```
tic()
nb_tune <- nb_workflow %>%
  tune_grid(resamples = cv_folds,
            grid = nb_grid,
            metrics = metric_set(accuracy, roc_auc, pr_auc, sens, yardstick::spec, pp
v, npv, f_meas),
            control = control_grid(verbose = TRUE))
toc()
```

1.1 Cross-validation metric distributions

In this section, we're going to take a little bit of a look at the individualized performance of the models taking into each fold into account. This will satisfy our academic curiosity in terms of machine learning and also provide some insight into the behaviors of the models. We'll look more at the aggregated measures in a moment.

We'll first decompress the tuning metrics a bit to get them into a more friendly form for processing.

Hide

```
#extract the cross validation metrics for the naive bayes model by fold (i.e., unsumma
rized)
nb_fold_metrics <- nb_tune %>%
  dplyr::select(id, .metrics, .notes) %>%
  unnest(.metrics)

head(nb_fold_metrics, 10)
```

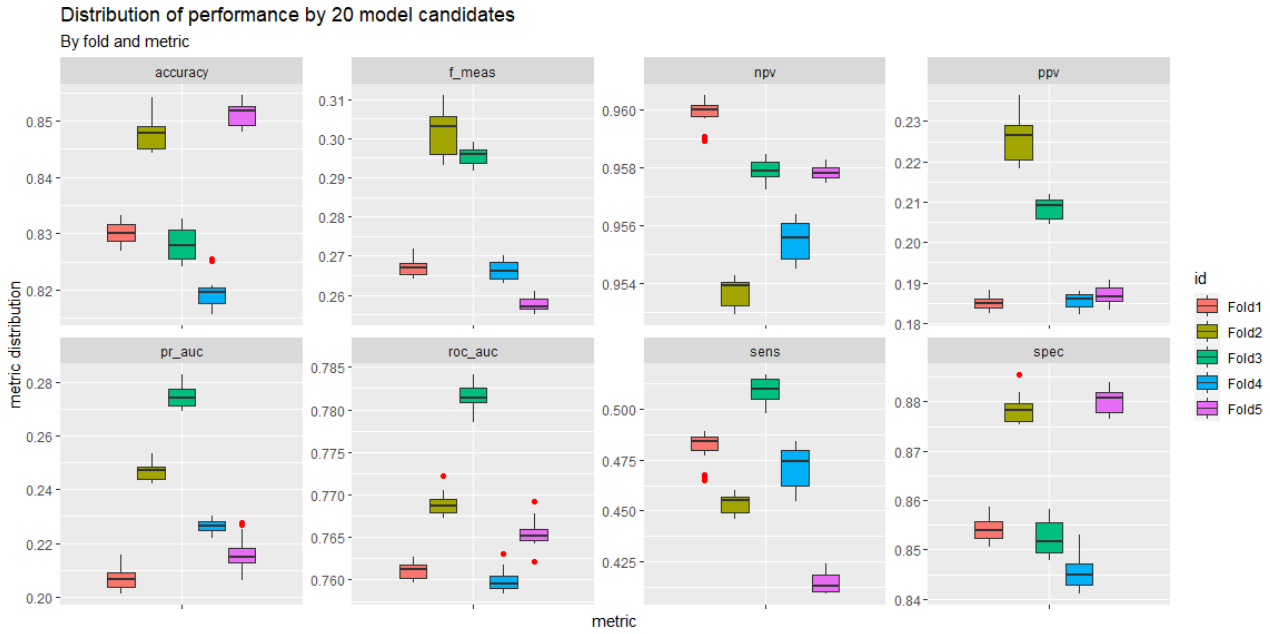
id	smoothn…	Laplace	.metric	.estimator	.estimate	.config
<chr>	<dbl>	<dbl>	<chr>	<chr>	<dbl>	<chr>
Fold1	1.4451107	2.938294	accuracy	binary	0.8331920	Preprocessor1_Model01
Fold1	1.4451107	2.938294	sens	binary	0.4677207	Preprocessor1_Model01
Fold1	1.4451107	2.938294	spec	binary	0.8583341	Preprocessor1_Model01
Fold1	1.4451107	2.938294	ppv	binary	0.1850886	Preprocessor1_Model01
Fold1	1.4451107	2.938294	npv	binary	0.9590845	Preprocessor1_Model01
Fold1	1.4451107	2.938294	f_meas	binary	0.2652223	Preprocessor1_Model01
Fold1	1.4451107	2.938294	roc_auc	binary	0.7599228	Preprocessor1_Model01
Fold1	1.4451107	2.938294	pr_auc	binary	0.2021701	Preprocessor1_Model01
Fold1	0.5841239	1.213577	accuracy	binary	0.8300543	Preprocessor1_Model02
Fold1	0.5841239	1.213577	sens	binary	0.4888011	Preprocessor1_Model02

1-10 of 10 rows

Now, let's visualize this generalized performance over all the models

Hide

```
nb_fold_metrics %>%
  mutate(facet_val = if_else(.metric== 'roc_auc' | .metric=='pr_auc' | .metric=='f_mea
s', 'Aggregate metrics', 'Confusion matrix metrics')) %>%
  ggplot(aes(x=.metric, y=.estimate, fill=id)) +
  geom_boxplot(outlier.colour = 'red', na.rm=TRUE) +
  facet_wrap(facet='.metric', scales='free', nrow=2) +
  labs(title='Distribution of performance by 20 model candidates',
       subtitle='By fold and metric',
       x='metric',
       y='metric distribution') +
  scale_x_discrete(labels=NULL)
```



Here we can see some interesting fold-specific results. There tended to be a large distubution for Fold 2 and fold 3 performance while Fold 3 and 1 were consistently small in terms of distribution.

2 Identifying the “best” model

Now, let’s collect the metrics to see how the model did over all of the folds and all of the metrics in order to identify the best model from these candidates. Note that this table looks similar to the prior tibble; the main difference here is that the results are aggregated over the folds (hence the `mean` and `n` columns).

Hide

```
tune_metrics <- nb_tune %>%
  collect_metrics()

head(tune_metrics, 5)
```

smoothn… <dbl>	Laplace <dbl>	.metric <chr>	.estimator <chr>	mean <dbl>	n <int>	std_err <dbl>	.config <chr>
1.445111	2.938294	accuracy	binary	0.8379385	5	0.005161482	Preprocessor1_
1.445111	2.938294	f_meas	binary	0.2760796	5	0.007684880	Preprocessor1_
1.445111	2.938294	npv	binary	0.9563047	5	0.001109958	Preprocessor1_
1.445111	2.938294	ppv	binary	0.1984149	5	0.006786374	Preprocessor1_
1.445111	2.938294	pr_auc	binary	0.2291511	5	0.012395085	Preprocessor1_

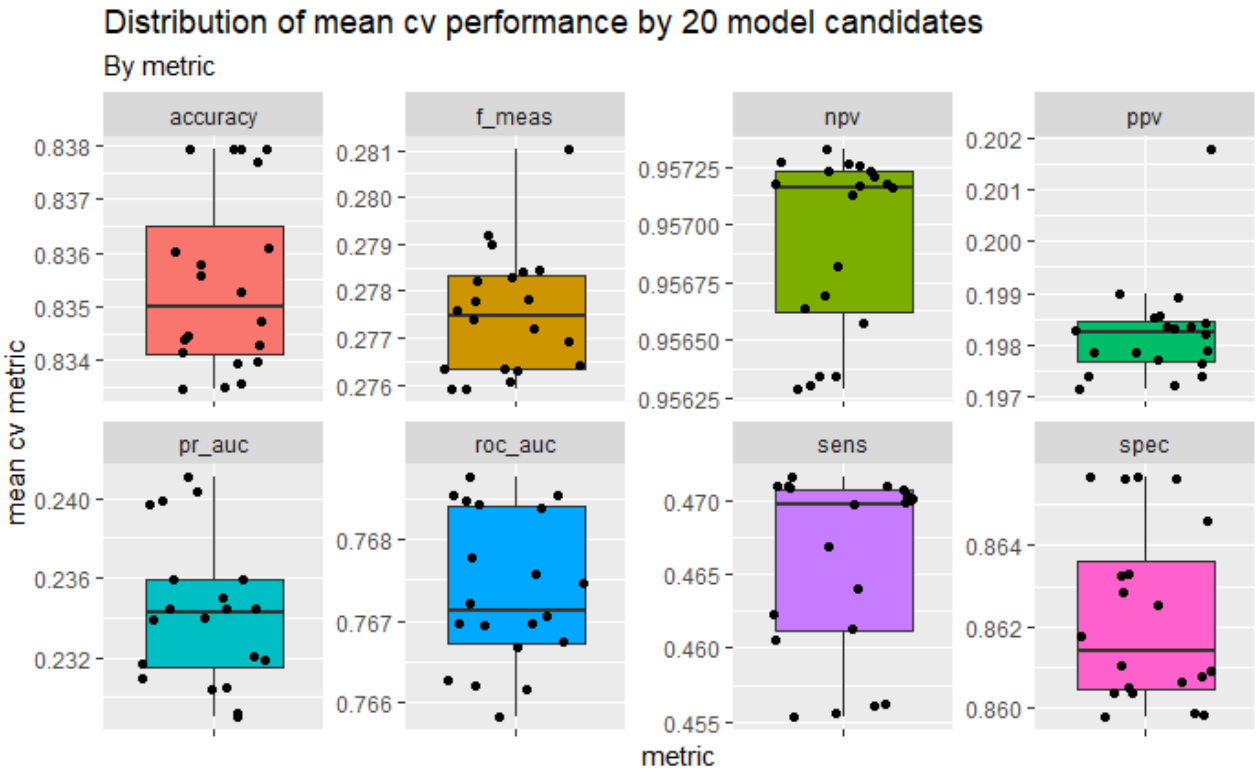
5 rows

2.1 Basic performance overview

Let's just look at the overall (fold-less) distribution of the metrics.

Hide

```
tune_metrics %>%
  ggplot(aes(x=.metric, y=mean)) +
  geom_boxplot(aes(fill=.metric), outlier.shape=NA, na.rm=TRUE) +
  geom_jitter(na.rm=TRUE) +
  facet_wrap(facets='.metric', nrow=2, scales='free') +
  theme(legend.position = 'none') +
  labs(title='Distribution of mean cv performance by 20 model candidates',
       subtitle='By metric',
       x='metric',
       y='mean cv metric') +
  scale_x_discrete(labels=NULL)
```



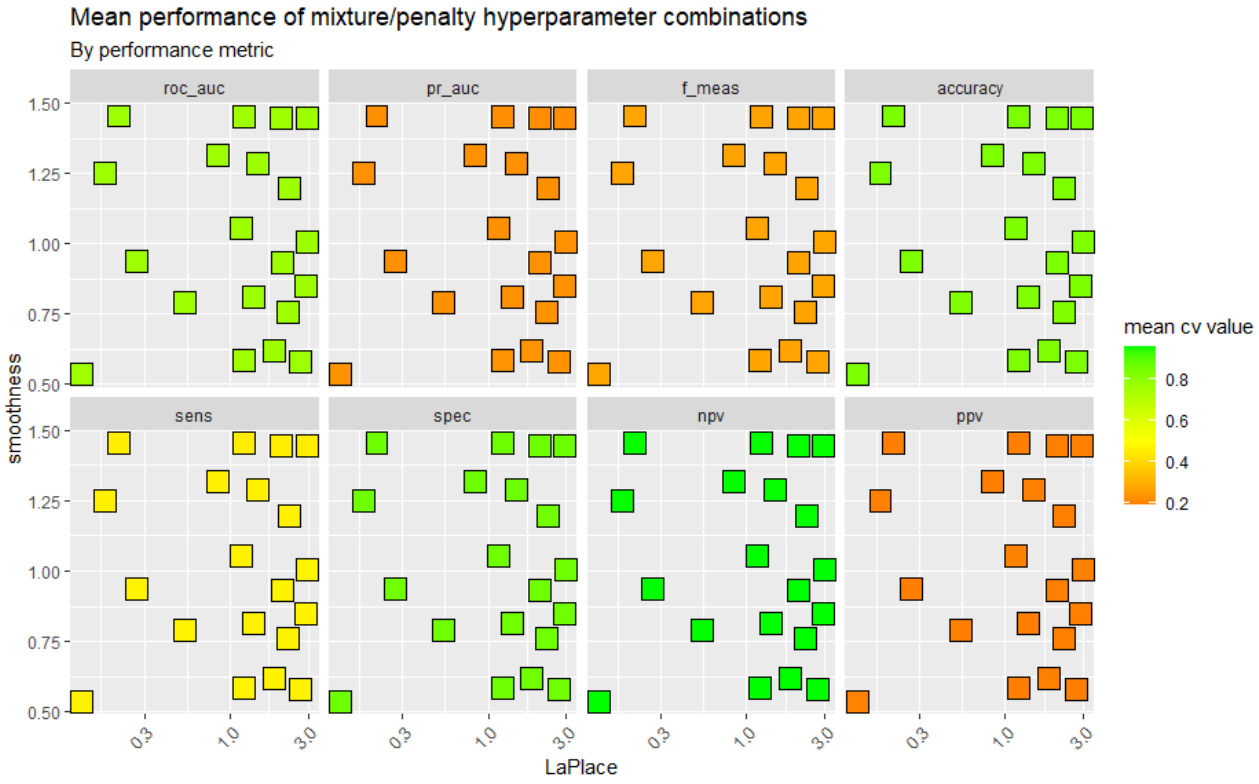
Here, we can see a relatively tight distribution for ppv and rather large distributions for spec, sense, and npv. However, all of these have a very small scale for their axises so that may be entirely normal. One additional observation is that for accuracy and pr_auc, there are several very high-performing outliers. These will likely be our candidates for the "best" model but them being outliers is an interesting area to be explored.

2.2 Making sense of the hyperparameters and their influence

Let's visualize this so we can make some sort of sense out of it.

Hide

```
tune_metrics %>%
  mutate(.metric=fct_relevel(.metric, 'roc_auc', 'pr_auc', 'f_meas', 'accuracy', 'sen
s', 'spec')) %>%
  ggplot(aes(x=Laplace, y=smoothness)) +
  geom_point(aes(fill=mean), shape=22, size=6) +
  scale_x_log10(guide=guide_axis(angle=45)) +
  facet_wrap(ncol=4, facet='.metric') +
  scale_fill_gradient2(low='red', mid='yellow', high='green', midpoint=0.5) +
  labs(title='Mean performance of mixture/penalty hyperparameter combinations',
       subtitle='By performance metric',
         x='LaPlace',
         y='smoothness',
         fill='mean cv value')
```



Here is a difficult-to-understand plot. The objective of this visualization is to begin to digest the relationship between the two hyperparameters and the performance given a certain metric. Recall that hyperparameter tuning evaluates all combinations of hyperparameters. These combos are shown as a square on a particular “subplot” of a metric of interest. Then, there are 20 squares since there are 20 models. And, the arrangement of all the “intersection” squares is identical.

What is of interest here is the color of the squares. Red indicates that the performance is poor, and green indicates that the performance is great. What is interesting here is that all of the squares within a given graph typically had the same color no matter the value of our two hyperparameters. This may suggest these hyperparameters don’t affect the cv value as much as the model itself does in terms of classification.

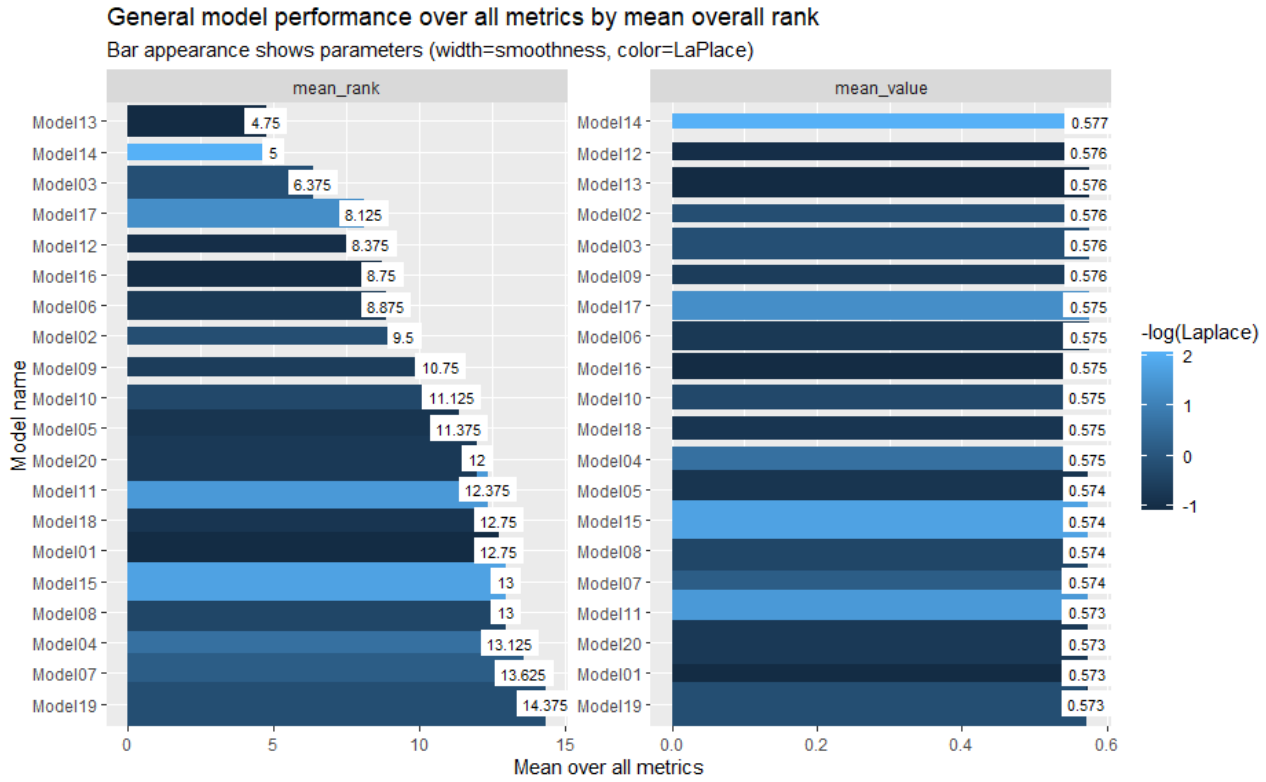
2.3 Looking at all of the metrics together to select a model

One possible way of evaluating a good model might be to rank the model according to its performance across all of the metrics. This allows us to get a bit away from the values themselves. However, we can also look at the values themselves and investigate the relationship.

Hide

```
#calculate mean metrics and rank
mdl_overall <- tune_metrics %>%
  group_by(.metric) %>%
  mutate(metr_rank=rank(-mean, ties.method='average')) %>% #-mean so that rank increas
es (so, worse) with decreasing metric
  group_by(.config, .add=FALSE) %>%
  mutate(mean_rank = mean(metr_rank)) %>% #add mean rank
  mutate(mean_value = -mean(mean, na.rm=TRUE)) %>% #add mean value
  pivot_longer(cols=c(mean_rank, mean_value), names_to = 'agg_perf_type', values_to='a
gg_perf') %>%
  group_by(agg_perf_type) %>%
  filter(.metric=='pr_auc') #just pick one set of values; all these aggregated values
 will be identical

#plot; note that there is manipulation of negatives for the directionality and absolut
e value
mdl_overall %>%
  ggplot(aes(x=reorder_within(str_remove(.config, 'Preprocessor1_'), -agg_perf, agg_pe
rf_type),
             y=abs(agg_perf),
             width=smoothness)) +
  geom_col(aes(fill=-log(Laplace))) +
  geom_label(aes(label=round(abs(agg_perf),3)), label.r=unit(0.0, "lines"), label.size
=0, size=3) +
  facet_wrap(~agg_perf_type, ncol=2, scales='free') +
  scale_x_reordered() +
  coord_flip() +
  labs(title='General model performance over all metrics by mean overall rank',
       subtitle='Bar appearance shows parameters (width=smoothness, color=LaPlace)',
       y='Mean over all metrics',
       x='Model name')
```



Here, the height of each bar represents smoothness while the color represents the Laplace parameter. Overall, we see skinnier bars, or those with a lower smoothness, perform better than those with a higher smoothness. Additionally, the visual color does not immediately seem to correlate with model performance with the Laplace.

2.4 Selecting the best model

With this information in mind as well as more help from tidymodels, we can then select the “best” model. One way to do this is to simply choose according to some metric. We’ll decide to use `pr_auc` here just because our training data is so imbalanced.

Hide

```
eval_metric <- 'pr_auc'

#show best parameters in terms of pr_auc
nb_tune %>% show_best(eval_metric)
```

smoothn…	Laplace	.metric	.estimator	mean	n	std_err	.config
<dbl>	<dbl>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
0.5347266	0.1248664	pr_auc	binary	0.2410878	5	0.01209749	Preprocessor1_M
0.5841239	1.2135775	pr_auc	binary	0.2403545	5	0.01224681	Preprocessor1_M
0.6182274	1.8503048	pr_auc	binary	0.2398425	5	0.01207038	Preprocessor1_M
0.5802238	2.7163315	pr_auc	binary	0.2396525	5	0.01220565	Preprocessor1_M
0.7585256	2.2693194	pr_auc	binary	0.2359066	5	0.01224579	Preprocessor1_M

5 rows

We find here that this is exactly in line with our previous assessment of overall model performance. Laplace does not seem to have much of an affect on the mean while the lower smoothness levels are very critical.

Hide

```
#select best parameters
best_nb_params <- nb_tune %>%
  select_best(eval_metric)

#show selected parameters
best_nb_params
```

smoothness	Laplace	.config
<dbl>	<dbl>	<chr>
0.5347266	0.1248664	Preprocessor1_Model14

1 row

We can see that Model 14 (best in overall rank and mean metric performance) predictably had the highest `pr_auc` .

3 Training fit

Having identified the best hyperparameters, we can create the final fit on all of the training data:

Hide

```
#finalize workflow with model hyperparameters
nb_final_wf <- nb_workflow %>%
  finalize_workflow(best_nb_params)
nb_final_wf
```

```
== Workflow ====================================================================
========================================
Preprocessor: Recipe
Model: naive_Bayes()

-- Preprocessor ----------------------------------------------------------------
----------------------------------------
2 Recipe Steps

* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
----------------------------------------
Naive Bayes Model Specification (classification)

Main Arguments:
  smoothness = 0.534726615762338
  Laplace = 0.124866351485252

Computational engine: naivebayes
```

Hide

```
#using final workflow, fit on training data
nb_final_fit <- nb_final_wf %>%
  fit(data = train_data)
```

4 Selected Model Performance Evaluation

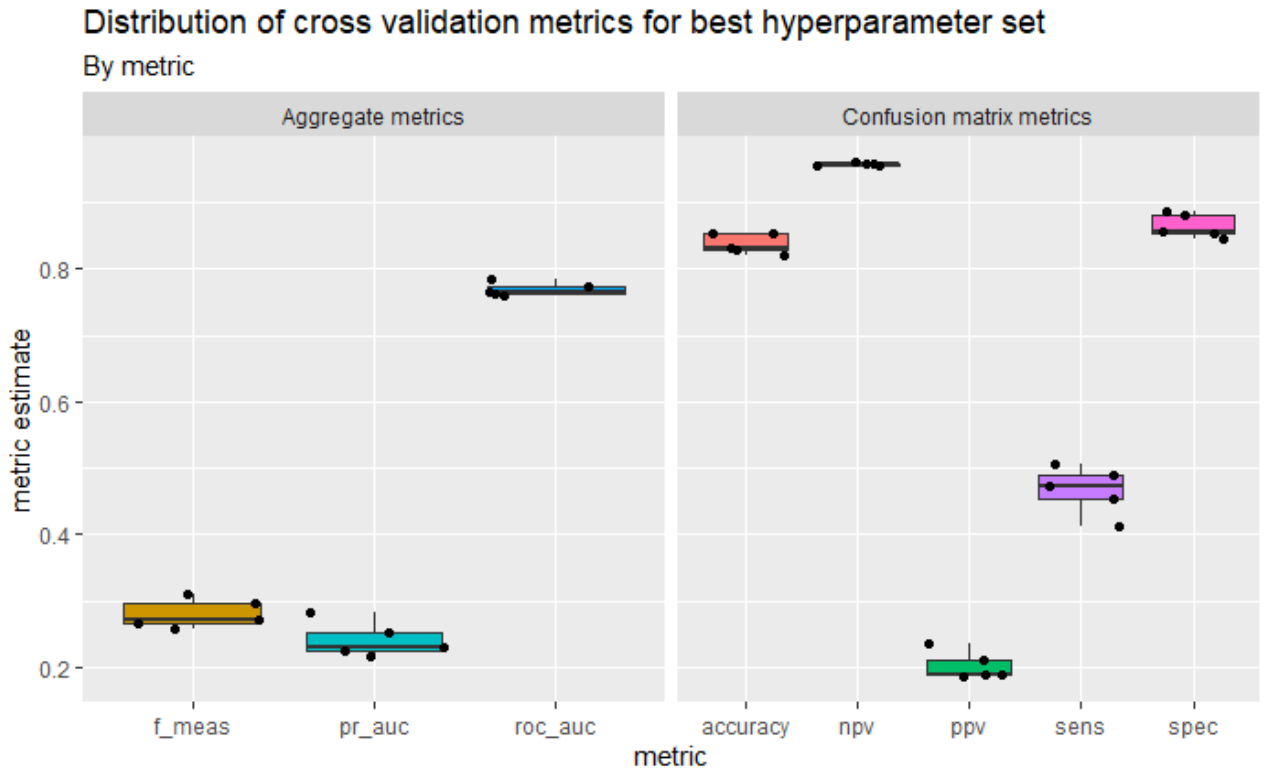
4.1 Cross validation metrics from best model

Let’s first evaluate the performance using the cross-validation metrics from before. However, here, we’ll only look at the best model.

Hide

```
#get best glmnet metrics
best_nb_fold_metrics <- nb_fold_metrics %>%
  filter(.config==best_nb_params$.config[[1]])

#plot
best_nb_fold_metrics %>%
  mutate(facet_val = if_else(.metric== 'roc_auc' | .metric=='pr_auc' | .metric=='f_mea
s', 'Aggregate metrics', 'Confusion matrix metrics')) %>%
  ggplot(aes(x=.metric, y=.estimate, fill=.metric)) +
  geom_boxplot(outlier.shape = NA, na.rm=TRUE) +
  geom_jitter(aes(x=.metric, y=.estimate), na.rm=TRUE) +
  facet_grid(cols=vars(facet_val), scales='free') + #just to get on separate plots
  labs(title='Distribution of cross validation metrics for best hyperparameter set',
       subtitle='By metric',
       x='metric',
       y='metric estimate') +
  theme(legend.position = "none")
```



We see very high values for accuracy, npv, and spec, all of which point to this model being one of the "outliers" we discussed earlier.

4.2 Performance on training data as a whole

Here, we look at the confusion matrix for the entire training set as well as computations from the confusion matrix.

Hide

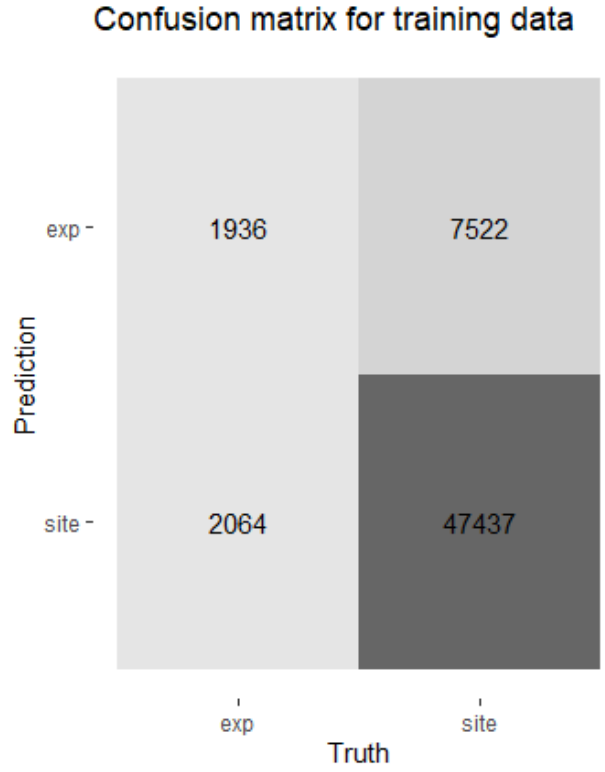
```
#get prediction class and probabilities
hp_training_preds <-
  predict(nb_final_fit, train_data) %>%
  bind_cols(predict(nb_final_fit, train_data, type = "prob")) %>%
  bind_cols(train_data %>%
              dplyr::select(particle_class))

#calculate confusion matrix
train_conf <- hp_training_preds %>%
  conf_mat(particle_class, .pred_class)

#get summary info
t1 <- train_conf %>%
  summary() %>%
  dplyr::select(-.estimator) %>%
  gridExtra::tableGrob(rows=NULL, theme=gridExtra::ttheme_default(base_size=10))

#plot cmat info
cm <- train_conf %>%
  autoplot(type='heatmap') +
  labs(title='Confusion matrix for training data')

gridExtra::grid.arrange(cm, t1, ncol=2)
```



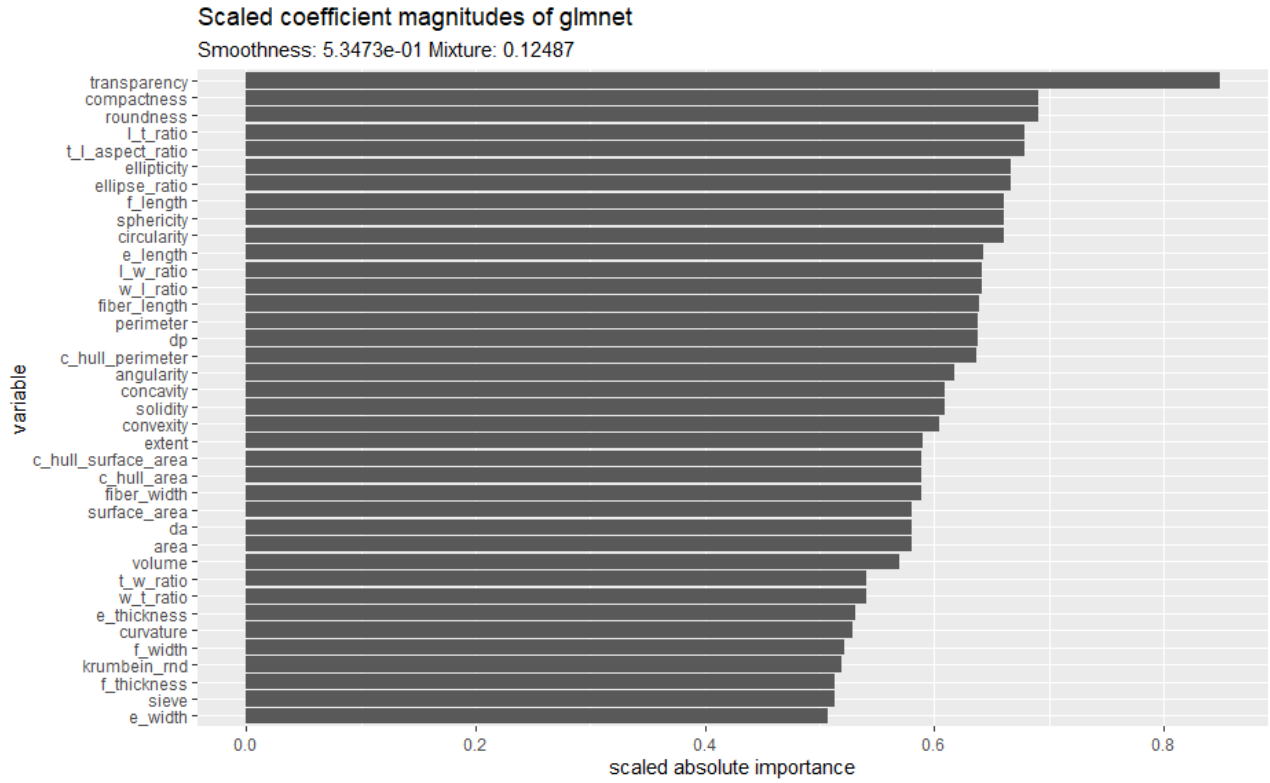
.metric	.estimate
accuracy	0.8374124
kap	0.2126274
sens	0.4840000
spec	0.8631343
ppv	0.2046944
npv	0.9583039
mcc	0.2378704
j_index	0.3471343
bal_accuracy	0.6735672
detection_prevalence	0.1604166
precision	0.2046944
recall	0.4840000
f_meas	0.2877099

Here we see there were roughly 7,500 particles that we truly site particles that were classified as lithic samples. The overall accuracy remains around 83.7% and precision at 20%.

5 Explaining the model

5.1 Variable imporance

What parameters are contributing most strongly to the classification? Do we see evidence of data snooping? Let's take a look!



Here we can clearly see a common thread through many of the models. The Naive bayes model also weighted transparency incredibly high followed by the typical compactness, roundness, and l_t_ratio which were also common candidates for high importance variables.

6 Save markdown file

Lastly, we'll just make sure to save this markdown file into the repo so that it may be easily accessed and viewed by everyone. To successfully use this, ***make sure you have saved your notebook and the .nb.html has been regenerated!!***

Hide

```
fs::file_copy('42-naivebayes-modeling.nb.html', './html_results/42-naivebayes-modelin
g.nb.html', overwrite=TRUE)
```