STA314 Assignment 2

Henry Shiffer 1003866881

Objective:

Classification is a predictive modeling problem where a class label is predicted based on a given input of data. This assignment shows how to use the logistic lasso model within a tidymodels workflow to perform classification. This logistic lasso model uses coordinate descent on the penalized iteratively reweighted least squares algorithm. After demonstrating the model, this assignment will show how to tune the model, and what to look for after tuning the model to make sure the model is working properly.

Getting started:

First, we will need to load `tidyverse` and `tidymodels` which are both needed to allow the logistic lasso model to run. We will then call `source("functions.R")` and `source("make_tidy.R")` which will call the code that contains the fit and predict function for the logistic lasso, and the code that registers the model to parsnip.

```
library(tidyverse)
library(tidymodels)
source("functions.R")
source("make_tidy.R")
```

Now we want to test our model's accuracy by looking at the confusion matrix. A confusion matrix is a tool that summarizes the preformance of a classification algorithm and will be useful to understand how the logistic lasso model preforms. To do this, we must first generate some random data. We will save this data as `henry_dat`. Next we must split our data into a training set and a test set. After this we set the engine of the `logistic_lasso` model that is registered in `"make_tidy.R"` to the model's fit function `fit_logistic_lasso` and train the model with the training set data. Finally, we can call the `predict()` function which then uses the testing set data to generate a confusion matrix for the `logistic_lasso` model.

```
#generate data 'henry_dat'
set.seed(3030)
n = 1000
henry_dat <- tibble(x = seq(-3,3, length.out = n),
w = 3*cos(3*seq(-pi,pi, length.out = n)),
y = rbinom(n,size = 1, prob = 1/(1 + exp(-w+2*x)) )%>% as.numeric %>% factor,
cat = sample(c("a","b","c"), n, replace = TRUE)
)
#splitting data into test set and training set
split <- initial_split(henry_dat, strata = c("cat"))
train <- training(split)
test <- testing(split)
rec <- recipe(y ~ . , data = train) %>%
step_dummy(all_nominal(), -y) %>% step_zv(all_outcomes()) %>%
step_normalize(all_numeric(), -y)

#setting the logistic_lasso model with the
#penalty 0.3 and adding the model to a workflow
```

```
spec <- logistic_lasso(penalty = 0.3) %>% set_engine("fit_logistic_lasso")
fit <- workflow() %>% add_recipe(rec) %>% add_model(spec) %>% fit(train)
predict(fit, new_data = test) %>%
  bind_cols(test %>% select(y)) %>%
  conf_mat(truth = y, estimate = .pred_class)
```

```
##            Truth
## Prediction   0   1
##          0 108   8
##          1  27 106
```

When looking at this confusion matrix we see that:

- The model predicted 0 and actually got 0, 108 times. This means that the model had 108 true negatives.
- The model predicted 1 and actually got 0, 27 times. This means that the model had 27 false positives.
- The model predicted 0 and actually got 1, 8 times. This means that the model had 8 false negatives.
- The model predicted 1 and actually got 1, 106 times. This means that the model had 106 true positives.

We see that it is a bit more likely to have a false positive than a false negative with this classification.

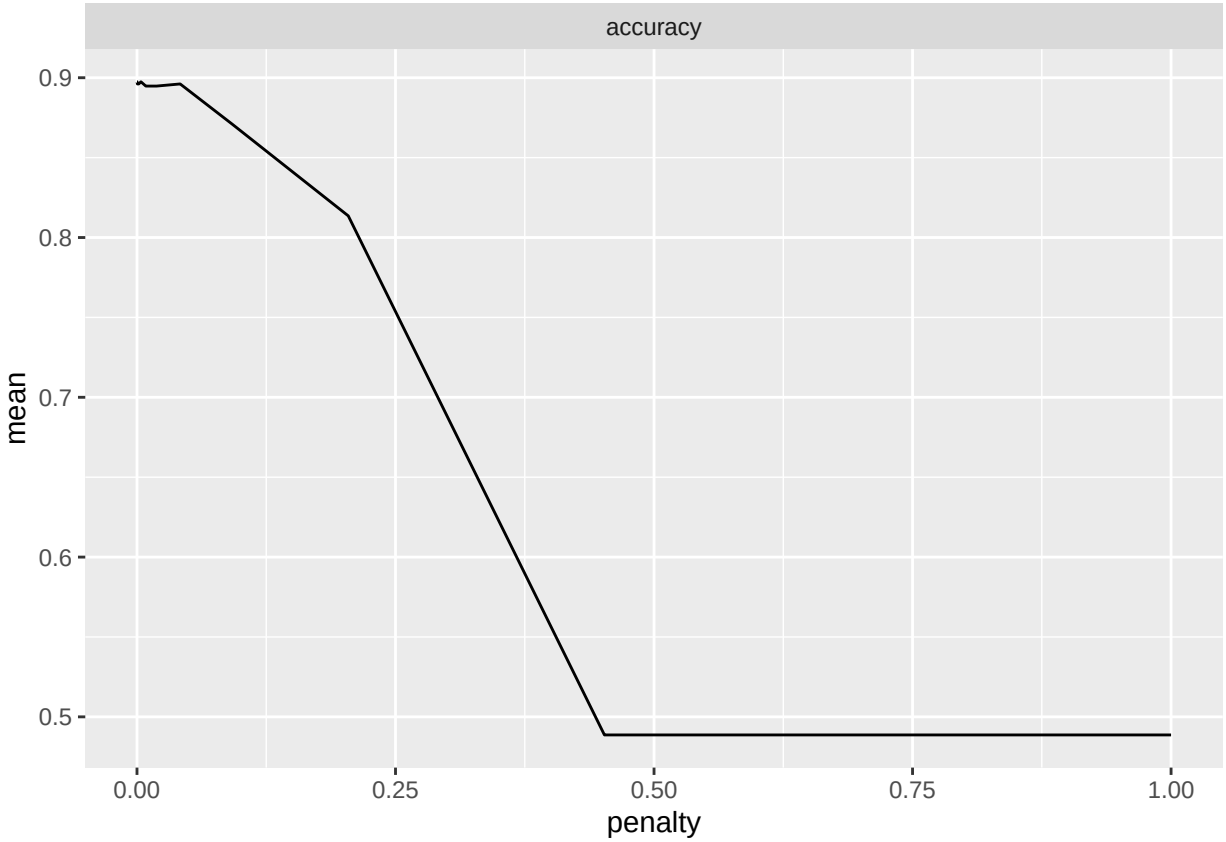
From here, we can see that the `logistic_lasso` model had a missclassification rate of: $(8 + 27) \ / \ 249 \sim 14\%$

We are going to want to see if we can decrease this missclassification rate by tuning the `logistic_lasso` model.

Tuning the model:

It is important now to tune the model, as tuning parameters of the model allows the model to preform better. First we can call `grid_regular` and set the levels to 30 to test 30 values of lambda. We can then tune the `logistic_lasso` penalty and add that to a workflow. After that, we are able to use cross validation to choose a value of lambda. Finally, we can tune the model and save it as `fit_tune`

```
grid <- grid_regular(penalty(), levels = 30)
spec_tune <- logistic_lasso(penalty = tune()) %>% set_engine("fit_logistic_lasso")
wf <- workflow() %>% add_recipe(rec) %>% add_model(spec_tune)
folds <- vfold_cv(train)
fit_tune <- wf %>% tune_grid(resamples = folds, grid = grid, metrics = metric_set(accuracy))
fit_tune %>% collect_metrics() %>% ggplot(aes(penalty, mean)) + geom_line() + facet_wrap(~.metric)
```



We can look at this graph and see that as the penalty grows to a little less than 0.5, the accuracy plateaus. This indicates that a lower lambda value may be better for the accuracy of the model.

Now we will use the function `select_best` on `fit_tune`. This selects the lambda that gives the best accuracy.

```
#selecting penalty with best accuracy
penalty_final <- fit_tune %>% select_best(metric = "accuracy")
#adding to the workflow
wf_final <- wf %>% finalize_workflow(penalty_final)
```

We can take a look now at the finalized workflow model specifications and see penalty that is suggested after the tune.

```
wf_final
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: logistic_lasso()
##
## -- Preprocessor ----------------------------------------------------------------
## 3 Recipe Steps
##
## * step_dummy()
## * step_zv()
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
## Model Specification (classification)
##
## Main Arguments:
```

```
##   penalty = 1e-10
##
## Computational engine: fit_logistic_lasso
```

Now that we have this final fit with the best value for lambda, we can run the function `predict()` to see if the confusion matrix has improved at all, which would indicate that the tuned model has preformed better.

```
final_fit <- wf_final %>% fit(train)
predict(final_fit, new_data = test) %>%
  bind_cols(test %>% select(y)) %>%
  conf_mat(truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction   0   1
##          0 126  11
##          1   9 103
```

Looking at this confusion matrix we see that:

- The model predicted 0 and actually got 0, 126 times. This means that the model had 126 true negatives.
- The model predicted 1 and actually got 0, 9 times. This means that the model had 9 false positives.
- The model predicted 0 and actually got 1, 11 times. This means that the model had 11 false negatives.
- The model predicted 1 and actually got 1, 103 times. This means that the model had 103 true positives.

Overall, this tuned confusion matrix is better than the previous confusion matrix. We see that the tuned `logistic_lasso` model has a missclassification rate of: $(9 + 11) / 249 \sim 8\%$. As this is lower than the missclassification rate of `logistic_lasso` before tuning, this indicates that the tuned model preforms better at classification.

Concluding remarks:

Throughout this vignette, it is shown how to interpret the classification confusion matrix, tune the 'logistic_lasso' model, select the best penalty for the model, and check if the model has improved at prediction. We have seen that the tuned `logistic_lasso` model has a smaller missclassification rate than before it was tuned which indicates that it has improved. Overall, logistic classification models are never perfect, and a more complex model may be needed when dealing with more complicated data.