Module 9 HW

Alanna Hazlett

2024-07-18

Module 9

> You can download the R Markdown file (https://gedeck.github.io/DS-6030/homework/Module-9.Rmd) and use it to answer the following questions.
>
> If not otherwise stated, use Tidyverse and Tidymodels for the assignments.

1. Sentiment analysis using SVM

In this assignment, we will build a model to predict the sentiment expressed in Amazon reviews. In order to build a model, we need to convert the text review into a numeric representation. We will use the `textrecipes` package to process the text data.

This assignment is only a first glimpse into handling text data. For a detailed introduction to text analytics in *tidymodels* see Hvitfeldt and Silge (2022, Supervised Machine Learning for Text Analysis in R).

The data are taken from https://archive.ics.uci.edu/dataset/331/sentiment+labelled+sentences.

You can load the data from https://gedeck.github.io/DS-6030/datasets/homework/sentiment_labelled_sentences/amazon_cells_labelled.txt

You will need to install the packages `textrecipes` and `stopwords` to complete this assignment.

(a.) Load the data. The data has no column headers. Each line contains a review sentence separated by a tab character (`\t`) from the sentiment label (0 or 1). Create a tibble with the column names `sentence` and `sentiment`. Use the *tidyverse* function `read_delim` to load the data. The dataset has 1000 rows. (the `read.csv` function fails to load the data correctly)

```
data<-read_delim(
"https://gedeck.github.io/DS-6030/datasets/homework/sentiment_labelled_sentences/amazon_cells_labelled.t
                 delim = "\t",
                 col_names=c('sentence','sentiment'),
                 show_col_types = FALSE)
data<-data %>%
  mutate(sentiment=as.factor(sentiment))
```

(b.) Split the dataset into training (80%) and test sets (20%). Prepare resamples from the training set for 10-fold cross validation.

```
set.seed(123)
split <- initial_split(data, prop=0.8, strata=sentiment)
train <- training(split)
holdout <- testing(split)

set.seed(123)
resamples <- vfold_cv(train, strata=sentiment)
```

```
cv_control <- control_resamples(save_pred=TRUE)
custom_metrics <- metric_set(roc_auc, j_index, accuracy)
```

(c.) Create a recipe to process the text data. The formula is `sentiment ~ sentence`. Add the following steps to the recipe: - `step_tokenize(sentence)` to tokenize the text (split into words). - `step_tokenfilter(sentence, max_tokens=1000)` to remove infrequent tokens keeping only the 1000 most frequent tokens. This will give you a term frequency matrix (for each token, how often a token occurs in the sentence) - `step_tfidf(sentence)` applies function to create a term frequency-inverse document frequency matrix. Use the `step_pca()` function to reduce the dimensionality of the data. For the PCA, you can either tune `num_comp` in a range of 200 to 700 or set it to a value of 400. Use the `step_normalize()` function to normalize the data.

```
formula <- sentiment ~ sentence
recipe<-recipe(formula, data=train) %>%
  step_tokenize(sentence) %>%
  step_tokenfilter(sentence, max_tokens=1000) %>%
  step_tfidf(sentence) %>%
  step_pca(num_comp=400) %>%
  step_normalize()
```

(d.) Create workflows with the recipe from (c) and tune the following models: Keep the default tuning ranges and only udpate `rbf_sigma` as mentioned. Use Bayesian hyperparameter optimization to tune the models. What are the tuned hyperparameters for each model?

- logistic regression with L1 regularization (`glmnet` engine tuning `penalty`)

```
logreg_spec <- logistic_reg(engine="glmnet", mode="classification",penalty=tune())
logreg_wf <- workflow() %>%
              add_recipe(recipe) %>%
              add_model(logreg_spec)
logreg_params <- extract_parameter_set_dials(logreg_wf) #%>%
    #update(
        #penalty=penalty(c(-2.5, -1.25)))
set.seed(123)
tune_results_logreg <- tune_bayes(logreg_wf,
                          resamples=resamples,
                          metrics=custom_metrics,
                          param_info=logreg_params,
                          iter=50)
```

```
## ! No improvement for 10 iterations; returning current results.
```

```
#n is number of top models we want
show_best(tune_results_logreg, metric='roc_auc', n=1)%>%
    select(penalty,.metric,mean,std_err,.iter) %>%
    knitr::kable(caption="Logistic Regression Best Model", digits=3)
```

Table 1: Logistic Regression Best Model

penalty	.metric	mean	std_err	.iter
0.019	roc_auc	0.878	0.015	3

```
#Get rid of scientific notation for plot
options(scipen=999)
autoplot(tune_results_logreg)
```

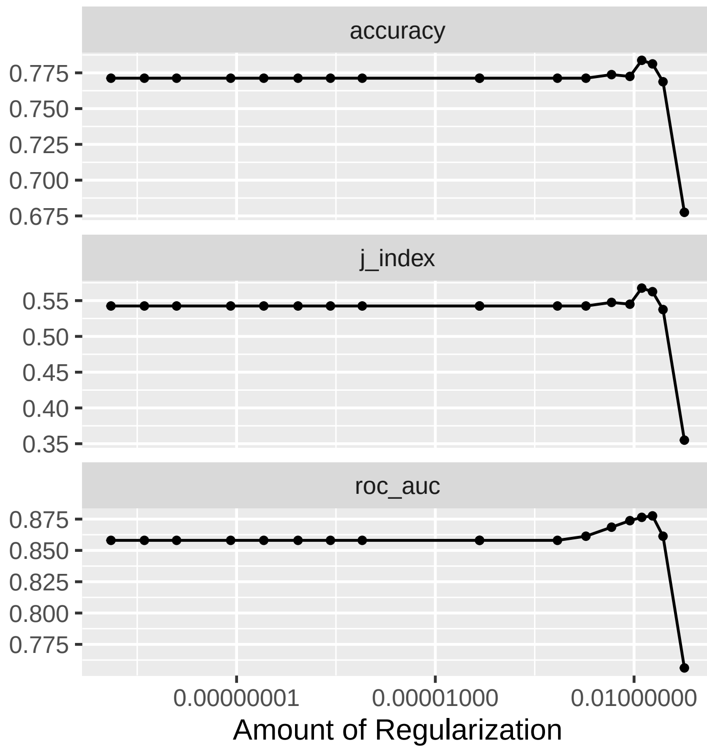

Figure 1: Logistic Regression Tuning Results

- SVM with linear kernel (`kernlab` engine tuning `cost` and `margin`)

```
lin_spec<-svm_linear(mode="classification", cost=tune(), margin=tune()) %>%
          set_engine("kernlab")
lin_wf<-workflow() %>%
        add_recipe(recipe) %>%
        add_model(lin_spec)
lin_params<-extract_parameter_set_dials(lin_wf)

set.seed(123)
tune_results_lin<-tune_bayes(lin_wf,
                             resamples=resamples,
                             metrics=custom_metrics,
                             param_info=lin_params,
                             iter=50)
```

```
## ! No improvement for 10 iterations; returning current results.
```

```
show_best(tune_results_lin, metric='roc_auc', n=1)%>%
    select(cost,margin,.metric,mean,std_err,.iter) %>%
    knitr::kable(caption="SVM Linear Best Model", digits=3)
```

Table 2: SVM Linear Best Model

cost	margin	.metric	mean	std_err	.iter
0.023	0.134	roc_auc	0.81	0.021	3

```r
options(scipen=999)
autoplot(tune_results_lin)
```

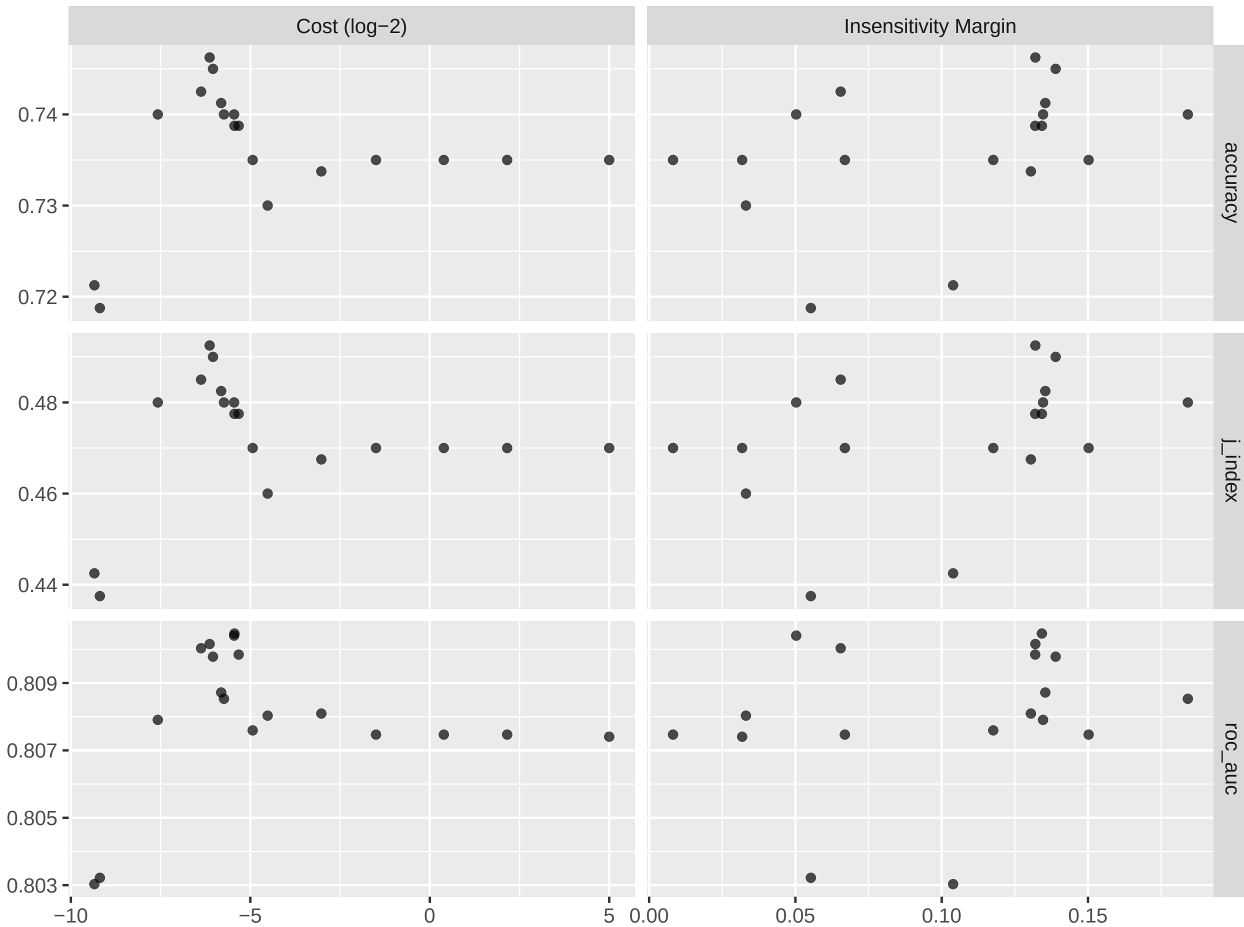

Figure 2: SVM Linear Tuning Results

- SVM with polynomial kernel (kernlab engine tuning cost, margin, and degree/degree_int)

```
poly_spec<-svm_poly(mode="classification", cost=tune(), margin=tune(), degree=tune()) %>%
                    set_engine("kernlab")
poly_wf<-workflow() %>%
        add_recipe(recipe) %>%
        add_model(poly_spec)
poly_params<-extract_parameter_set_dials(poly_wf)

set.seed(123)
tune_results_poly<-tune_bayes(poly_wf,
                             resamples=resamples,
                             metrics=custom_metrics,
                             param_info=poly_params,
                             iter=50)
```

```
## ! No improvement for 10 iterations; returning current results.
```

```
show_best(tune_results_poly, metric='roc_auc', n=1)%>%
    select(cost,margin,degree,.metric,mean,std_err,.iter) %>%
    knitr::kable(caption="SVM Polynomial Best Model", digits=3)
```

Table 3: SVM Polynomial Best Model

cost	margin	degree	.metric	mean	std_err	.iter
0.013	0.002	1	roc_auc	0.811	0.022	6

```
options(scipen=999)
autoplot(tune_results_poly)
```

- SVM with radial basis function kernel (`kernlab` engine tuning `cost`, `margin`, and `rbf_sigma`; use `rbf_sigma(range=c(-4, 0), trans=log10_trans())`)

```
rbf_spec<-svm_rbf(mode="classification", cost=tune(), margin=tune(), rbf_sigma=tune()) %>%
                    set_engine("kernlab")
rbf_wf<-workflow() %>%
        add_recipe(recipe) %>%
        add_model(rbf_spec)
rbf_params<-extract_parameter_set_dials(rbf_wf) %>%
              update(rbf_sigma=rbf_sigma(c(-4, 0),trans=log10_trans()))

set.seed(123)
tune_results_rbf<-tune_bayes(rbf_wf,
                             resamples=resamples,
                             metrics=custom_metrics,
                             param_info=rbf_params,
                             iter=50)
```

```
## ! No improvement for 10 iterations; returning current results.
```

```
show_best(tune_results_rbf, metric='roc_auc', n=1)%>%
    select(cost,margin,rbf_sigma,.metric,mean,std_err,.iter) %>%
    knitr::kable(caption="SVM RBF Best Model", digits=3)
```

Table 4: SVM RBF Best Model

cost	margin	rbf_sigma	.metric	mean	std_err	.iter
26.063	0.036	0	roc_auc	0.807	0.02	19

```
options(scipen=999)
autoplot(tune_results_rbf)
```

(e.) Once you have tuned the models, fit finalized models and assess their performance.

```
finalized_logreg_wf<- logreg_wf %>%
                      finalize_workflow(select_best(tune_results_logreg, metric="roc_auc"))
logreg_model <- finalized_logreg_wf%>%
                  fit(train)
finalized_lin_wf<-lin_wf %>%
                  finalize_workflow(select_best(tune_results_lin, metric="roc_auc"))
lin_model<- finalized_lin_wf%>%
              fit(train)
```

```
##  Setting default kernel parameters
```

```
finalized_poly_wf<-poly_wf %>%
                  finalize_workflow(select_best(tune_results_poly, metric="roc_auc"))
poly_model<- finalized_poly_wf%>%
            fit(train)
finalized_rbf_wf<-rbf_wf %>%
                  finalize_workflow(select_best(tune_results_rbf, metric="roc_auc"))
rbf_model<- finalized_rbf_wf%>%
            fit(train)
```

(i.) Compare the cross-validation performance of the models using ROC curves and performance metrics (AUC and accuracy). Which model performs best?

```
logreg_fit_cv <- finalized_logreg_wf %>%
              fit_resamples(resamples, control=cv_control,metrics=custom_metrics)
lin_fit_cv <- finalized_lin_wf %>%
              fit_resamples(resamples, control=cv_control,metrics=custom_metrics)
poly_fit_cv <- finalized_poly_wf %>%
              fit_resamples(resamples, control=cv_control,metrics=custom_metrics)
rbf_fit_cv <- finalized_rbf_wf %>%
              fit_resamples(resamples, control=cv_control,metrics=custom_metrics)

#Make table of CV performance metrics
cv_metrics<-bind_rows(
    collect_metrics(logreg_fit_cv) %>% mutate(model="Logistic Regression"),
    collect_metrics(lin_fit_cv) %>% mutate(model="SVM Linear"),
    collect_metrics(poly_fit_cv) %>% mutate(model="SVM Polynomial"),
    collect_metrics(rbf_fit_cv) %>% mutate(model="SVM RBF"),
)

cv_metrics%>%
    select(model, .metric, mean) %>%
    pivot_wider(names_from = .metric, values_from = mean) %>%
    knitr::kable(caption="Cross-validation performance metrics", digits=3)
```

Table 5: Cross-validation performance metrics

model	accuracy	j_index	roc_auc
Logistic Regression	0.781	0.562	0.878
SVM Linear	0.739	0.478	0.810
SVM Polynomial	0.745	0.490	0.811
SVM RBF	0.731	0.462	0.807

```
ggplot(cv_metrics, aes(x=mean, y=model, xmin=mean-std_err, xmax=mean+std_err)) +
    geom_point() +
    geom_linerange() +
    facet_wrap(~ .metric)+
    labs(title="Cross Validation Metrics Across Models")
```

```
roc_cv_data <- function(model_cv) {
    cv_predictions <- collect_predictions(model_cv)
    cv_predictions %>%
        roc_curve(truth=sentiment, .pred_1, event_level="second")
}
g1 = bind_rows(
    roc_cv_data(logreg_fit_cv) %>% mutate(model="Logistic regression"),
    roc_cv_data(lin_fit_cv) %>% mutate(model="Linear SVM"),
    roc_cv_data(poly_fit_cv) %>% mutate(model="Polynomial SVM"),
    roc_cv_data(rbf_fit_cv) %>% mutate(model="RBF SVM")
) %>%
ggplot(aes(x=1-specificity, y=sensitivity, color=model)) +
    geom_line()+
    labs(title="CV ROC AUC")

g1
```

The cross validation metrics from Figure 5 and Table 5 show that Logistic Regression with L1 regularization best represents our data as it has the highest values for accuracy and AUC. It also has the highest value for J index as well.

(ii.) Compare the performance of the finalized models on the test set. Which model performs best?

```
my_metrics=metric_set(j_index, accuracy)
tnt_metrics<-bind_rows(
    my_metrics(augment(logreg_model, holdout),truth= sentiment,
                  estimate=.pred_class,event_level="first") %>%
                        mutate(model="Logistic Regression",dataset="Test"),
    my_metrics(augment(lin_model, holdout),truth= sentiment,
                  estimate=.pred_class,event_level="first") %>%
                        mutate(model="SVM Linear",dataset="Test"),
    my_metrics(augment(poly_model, holdout),truth= sentiment,
                  estimate=.pred_class,event_level="first") %>%
                        mutate(model="SVM Poly",dataset="Test"),
    my_metrics(augment(rbf_model, holdout),truth= sentiment,
                  estimate=.pred_class,event_level="first") %>%
                        mutate(model="SVM RBF",dataset="Test"),
)


tnt_metrics%>%
    select(model, dataset, .metric, .estimate) %>%
    pivot_wider(names_from = .metric, values_from = .estimate) %>%
    knitr::kable(caption="Performance metrics", digits=3)
```

Table 6: Performance metrics

model	dataset	j_index	accuracy
Logistic Regression	Test	0.60	0.800
SVM Linear	Test	0.45	0.725
SVM Poly	Test	0.49	0.745
SVM RBF	Test	0.51	0.755

```
logreg_auc<-roc_auc(logreg_model %>% augment(holdout), sentiment, .pred_1, event_level="second")
lin_auc<-roc_auc(lin_model %>% augment(holdout), sentiment, .pred_1, event_level="second")
poly_auc<-roc_auc(poly_model %>% augment(holdout), sentiment, .pred_1, event_level="second")
rbf_auc<-roc_auc(rbf_model %>% augment(holdout), sentiment, .pred_1, event_level="second")

auc_metrics<-bind_rows(logreg_auc%>%
          mutate(model="Logistic Regression",dataset="Test"),

          lin_auc%>%
          mutate(model="SVM Linear",dataset="Test"),

          poly_auc%>%
          mutate(model="SVM Poly",dataset="Test"),

          rbf_auc%>%
          mutate(model="SVM RBF",dataset="Test"))

auc_metrics%>%
```

```
select(model, dataset, .metric, .estimate) %>%
pivot_wider(names_from = .metric, values_from = .estimate) %>%
knitr::kable(caption="Performance metrics", digits=3)
```

Table 7: Performance metrics

model	dataset	roc_auc
Logistic Regression	Test	0.885
SVM Linear	Test	0.806
SVM Poly	Test	0.808
SVM RBF	Test	0.827

In Tables 6 and 7 we see that the Logistic Regression Model with L1 Regularization outperforms the other models. It has the best (highest) values for accuracy, j index, and AUC.

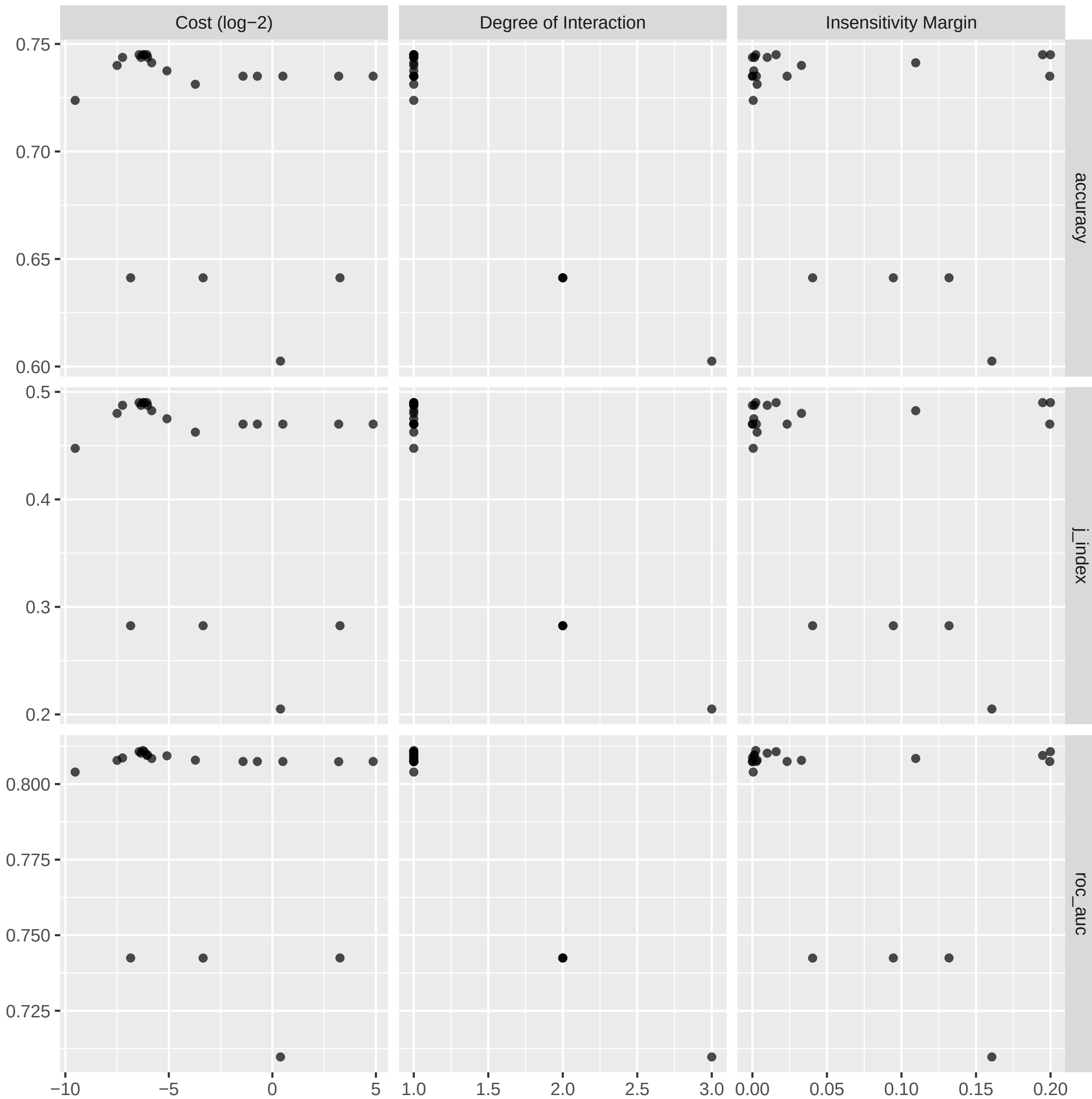

Figure 3: SVM Polynomial Tuning Results

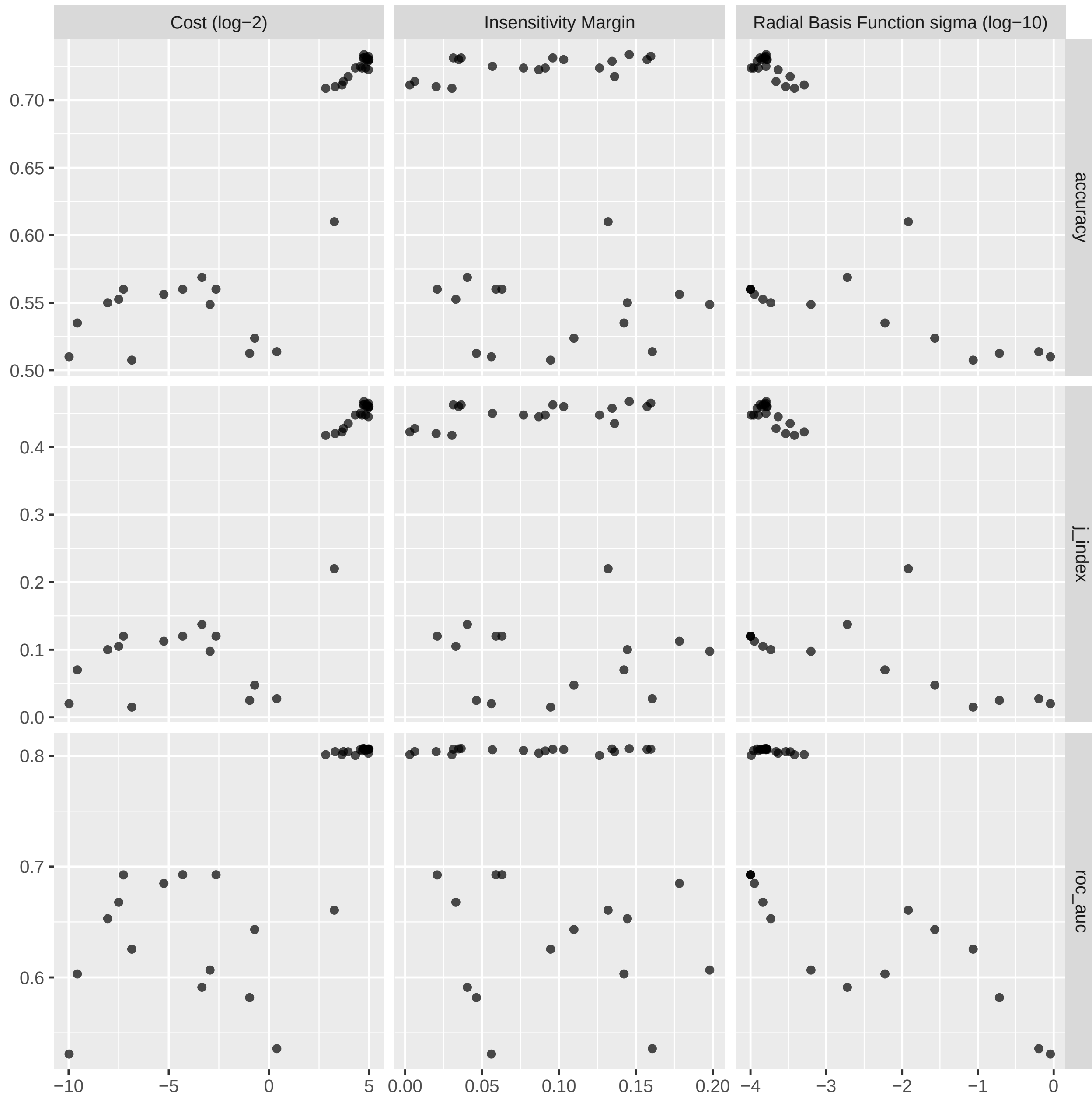

Figure 4: Logistic Regression Tuning Results

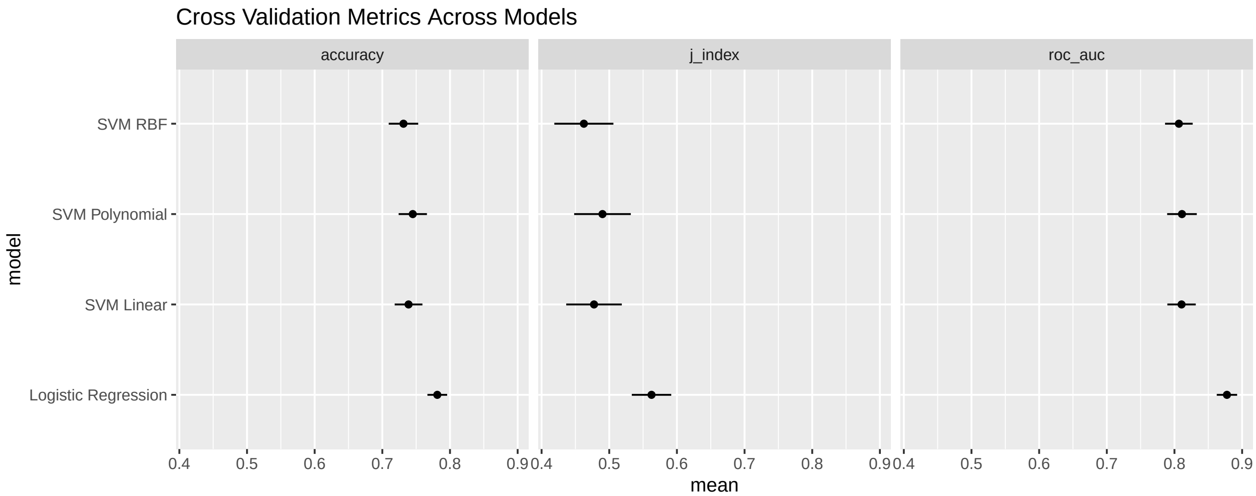

Figure 5: CV Metrics

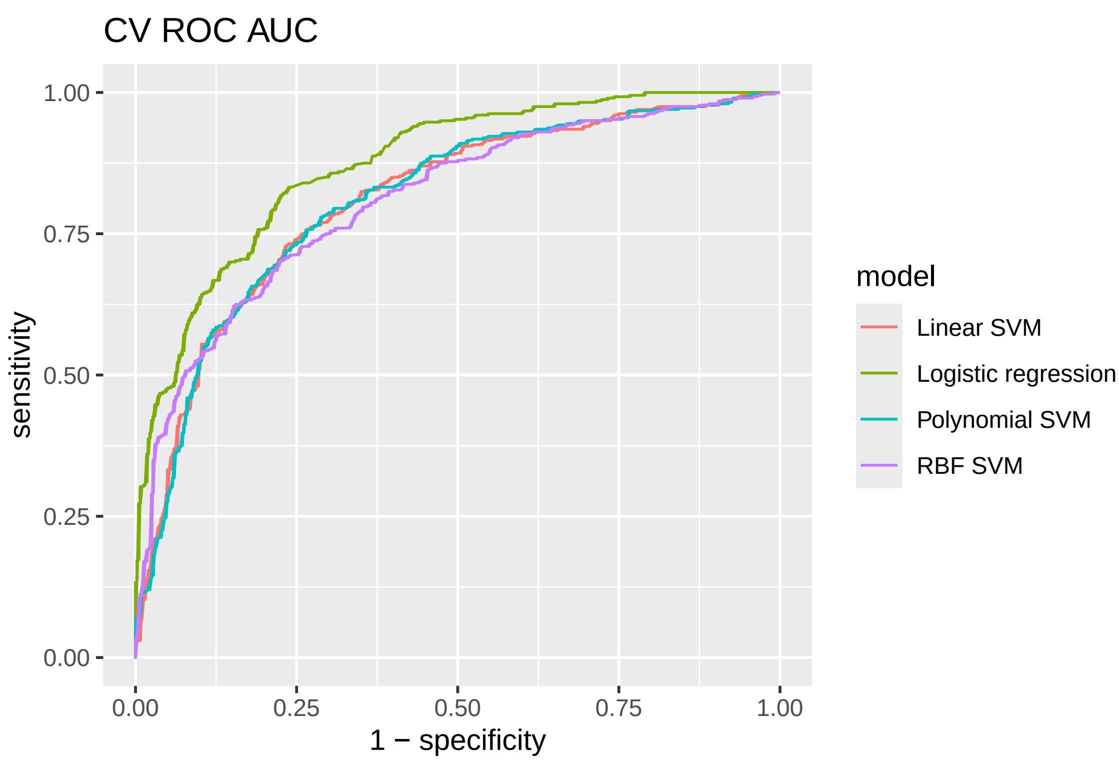

Figure 6: CV ROC