Thesis Simulation Single Run for Chapter 4

Dasha Asienga

2024-03-27

Contents

This file is intended to run the simulation process on the parent simulation data set as a single run, before scaling it into multiple trials.

Logistic Regression

Reading in the Data

First, let's read in the parent simulation data set.

```
compas_sim_path <- "/home/dasienga24/Statistics-Senior-Honors-Thesis/Data Sets/COMPAS/compas_sim.csv"
compas_sim_parent <- read.csv(compas_sim_path)
```

Fitting the Logistic Regression

We'll want to run the logistic regression as our baseline model and obtain the 3 key performance measures: convergence, accuracy, and discrimination.

```
lr <- glm(is_recid ~ age + prior_offense,
            data = compas_sim_parent,
            family = binomial(logit))
```

Convergence

Obtain convergence as an object. We would expect LR to always converge.

```
lr_converged <- lr[["converged"]]
lr_converged
```

```
## [1] TRUE
```

Accuracy

Conditional on convergence, obtain the accuracy as an object, which in this case is 77.96%.

```
lr_accuracy <- count(round(lr[["fitted.values"]]) == lr[["y"]])/nrow(compas_sim_parent)
lr_accuracy
```

```
## n_TRUE
## 0.7796
```

Discrimination

Conditional on convergence, obtain the discrimination statistic as an object, which in this case is 0.2464 or 24.64%.

```
preds <- predict(lr, newdata = compas_sim_parent, type="response")

compas_sim_parent <- compas_sim_parent %>%
  mutate(preds = preds,
         prediction = round(preds, 0),
         pred_risk = ifelse(prediction == 0, 'Low', 'High'))

discrimination <- compas_sim_parent %>%
  dplyr::select(race, pred_risk, is_recid) %>%
  group_by(race, is_recid) %>%
  mutate(total = n()) %>%
  group_by(pred_risk, race, total) %>%
  summarise("reoffended" = count(is_recid == 1),
            "did_not_reoffend" = count(is_recid == 0)) %>%
  pivot_longer(cols = c("reoffended", "did_not_reoffend"),
               names_to = "recidivism") %>%
  pivot_wider(
    id_cols = c("pred_risk", "recidivism", "total"),
    names_from = "race",
    values_from = value
  ) %>%
  rename("Black" = `African-American`,
         "White" = `Caucasian`) %>%
  mutate(Black = round(100 * Black / total, 2),
```

```r
         White = round(100 * White / total, 2)) %>%
  dplyr::select(-total) %>%
  group_by(pred_risk, recidivism) %>%
  summarize(Black = max(Black, na.rm = TRUE),
            White = max(White, na.rm = TRUE)) %>%
  filter((pred_risk == "High" & recidivism == "did_not_reoffend") |
           (pred_risk == "Low" & recidivism == "reoffended")
  )
```

```r
lr_disc_stat <- sum(abs(discrimination$White - discrimination$Black))/100
lr_disc_stat
```

```
## [1] 0.2464
```

The results from the logistic regression are now easily savable and retrievable, which will be useful when we scale the process.

Seldonian Framework

We also want to be able to easily retrieve the 3 key performance measures from the Seldonian algorithms we run before we scale the process, that is, convergence, accuracy, and discrimination.

Reading in the Data

First, let's read in the data.

```python
# point to the data file
f_orig = "/home/dasienga24/Statistics-Senior-Honors-Thesis/Data Sets/COMPAS/compas_sim.csv"

columns_orig = ["race","prior_offense","age", "is_recid"]

df = pd.read_csv(f_orig, header=0, names=columns_orig)
```

Pre-Processing the Data

Let's also preprocess the data to prepare it for Seldonian modeling.

```python
# select inputs to be transformed
X = df.drop(columns=["is_recid"])
y = df["is_recid"]

# one hot encode race, scale age using standard scaler
ct = ColumnTransformer([('c',OneHotEncoder(),['race']), ('n',StandardScaler(),['age'])])

# apply transformation
X_transformed = ct.fit_transform(X)

# get names after one-hot encoding
output_columns = ct.get_feature_names_out(ct.feature_names_in_)

# make an output dataframe to save transformed X and y
outdf = pd.DataFrame(X_transformed,columns=output_columns)
```

```python
# change names of columns
outdf.rename(columns={'c__race_African-American':'Black', 'c__race_Caucasian':'White', 'n__age':'age'},
```

```python
# re-index in order to concatenate columns
prior_offense = df["prior_offense"]
y.index = range(0, len(y))
prior_offense.index = range(0, len(prior_offense))
```

```python
# add label column and `prior_offense` into final dataframe
outdf['prior_offense'] = prior_offense
outdf['is_recid'] = y
```

The data set is now clean and ready. Let's save it along with the JSON metadata file.

```python
# save final dataframe
output_path_data="/home/dasienga24/Statistics-Senior-Honors-Thesis/Data Sets/COMPAS/Simulation Single R

outdf.to_csv(output_path_data,index=False,header=False)
```

```python
# save metadata json file
output_path_metadata="/home/dasienga24/Statistics-Senior-Honors-Thesis/Data Sets/COMPAS/Simulation Singl

metadata_dict = {
  "regime":"supervised_learning",
  "sub_regime":"classification",
  "all_col_names":list(outdf.columns),
  "label_col_names":"is_recid",
  "sensitive_col_names":["Black", "White"]
}

with open(output_path_metadata,'w') as outfile:
    json.dump(metadata_dict,outfile,indent=2)
```

Fitting a Seldonian Algorithm

Varying $\epsilon = 0.2, 0.1, 0.05, \& \ 0.01$, let's fit a Seldonian algorithm such that

$$abs((FNR|[Black]) - (FNR|[White])) + abs((FPR|[Black]) - (FPR|[White])) <= \epsilon.$$

We will take $\delta = 0.05$ to ensure 95% confidence.

First, let's read in the data set and specify the regime.

```python
import autograd.numpy as np

data_pth = output_path_data
metadata_pth = output_path_metadata
save_dir = "/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/"
os.makedirs(save_dir,exist_ok=True)

# create dataset from data and metadata file
```

```
regime='supervised_learning'
sub_regime='classification'

loader = DataSetLoader(regime=regime)

dataset = loader.load_supervised_dataset(
  filename=data_pth,
  metadata_filename=metadata_pth,
  file_type='csv')

sensitive_col_names = dataset.meta.sensitive_col_names

# use logistic regression model
model = LogisticRegressionModel()

# set the primary objective to be log loss
primary_objective = objectives.binary_logistic_loss
```

Next, let's create and save the specification files for each of the four levels of ϵ.

```
from seldonian.spec import createSupervisedSpec

# define behavioral constraints (epsilon = 0.2)
epsilon = 0.2
constraint_name = "equalized_odds"
if constraint_name == "equalized_odds":
  constraint_strs = [f'abs((FNR | [Black]) - (FNR | [White])) + abs((FPR | [Black]) - (FPR | [White]))
deltas = [0.05]

# create spec file
save_dir = "/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/equali

os.makedirs(save_dir, exist_ok=True) #create folder

createSupervisedSpec(
            dataset=dataset,
            metadata_pth=metadata_pth,
            constraint_strs=constraint_strs,
            deltas=deltas,
            save_dir=save_dir,
            save=True,
            verbose=False)
```

```
## <seldonian.spec.SupervisedSpec object at 0x7ffe8d7eb790>
```

```
#--------------------------------------------#

# define behavioral constraints (epsilon = 0.1)
epsilon = 0.1
constraint_name = "equalized_odds"
if constraint_name == "equalized_odds":
  constraint_strs = [f'abs((FNR | [Black]) - (FNR | [White])) + abs((FPR | [Black]) - (FPR | [White]))
```

```
deltas = [0.05]

# create spec file
save_dir = "/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/equali

os.makedirs(save_dir, exist_ok=True) #create folder

createSupervisedSpec(
            dataset=dataset,
            metadata_pth=metadata_pth,
            constraint_strs=constraint_strs,
            deltas=deltas,
            save_dir=save_dir,
            save=True,
            verbose=False)
```

```
## <seldonian.spec.SupervisedSpec object at 0x7ffe8d926bd0>
```

```
#-----------------------------------------------#

# define behavioral constraints (epsilon = 0.05)
epsilon = 0.05
constraint_name = "equalized_odds"
if constraint_name == "equalized_odds":
  constraint_strs = [f'abs((FNR | [Black]) - (FNR | [White])) + abs((FPR | [Black]) - (FPR | [White]))
deltas = [0.05]

# create spec file
save_dir = "/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/equali

os.makedirs(save_dir, exist_ok=True) #create folder

createSupervisedSpec(
            dataset=dataset,
            metadata_pth=metadata_pth,
            constraint_strs=constraint_strs,
            deltas=deltas,
            save_dir=save_dir,
            save=True,
            verbose=False)
```

```
## <seldonian.spec.SupervisedSpec object at 0x7ffe8d7edfd0>
```

```
#----------------------------------------------#

# define behavioral constraints (epsilon = 0.01)
epsilon = 0.01
constraint_name = "equalized_odds"
if constraint_name == "equalized_odds":
  constraint_strs = [f'abs((FNR | [Black]) - (FNR | [White])) + abs((FPR | [Black]) - (FPR | [White]))
deltas = [0.05]

# create spec file
save_dir = "/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/equali
```

```python
os.makedirs(save_dir, exist_ok=True) #create folder

createSupervisedSpec(
            dataset=dataset,
            metadata_pth=metadata_pth,
            constraint_strs=constraint_strs,
            deltas=deltas,
            save_dir=save_dir,
            save=True,
            verbose=False)
```

```
## <seldonian.spec.SupervisedSpec object at 0x7ffe8d9beb10>
```

Finally, let's run the Seldonian engine for each of the four specification files.

```python
from seldonian.seldonian_algorithm import SeldonianAlgorithm
from seldonian.utils.io_utils import load_pickle

# load the spec file (epsilon = 0.2)
specfile = '/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/equali
spec = load_pickle(specfile)
SA_02 = SeldonianAlgorithm(spec)

#-----------------------------------#

# load the spec file (epsilon = 0.1)
specfile = '/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/equali
spec = load_pickle(specfile)
SA_01 = SeldonianAlgorithm(spec)

#-----------------------------------#

# load the spec file (epsilon = 0.05)
specfile = '/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/equali
spec = load_pickle(specfile)
SA_005 = SeldonianAlgorithm(spec)

#-----------------------------------#

# load the spec file (epsilon = 0.01)
specfile = '/home/dasienga24/Statistics-Senior-Honors-Thesis/Python/COMPAS Simulation/Single Run/equali
spec = load_pickle(specfile)
SA_001 = SeldonianAlgorithm(spec)
```

Convergence

Obtain and store convergence as an object.

epsilon = 0.2

```python
passed_safety_02, solution_02 = SA_02.run(write_cs_logfile=True)
passed_safety_02
```

```
## True
```

epsilon = 0.1

```
passed_safety_01, solution_01 = SA_01.run(write_cs_logfile=True)
passed_safety_01
```

```
## True
```

epsilon = 0.05

```
passed_safety_005, solution_005 = SA_005.run(write_cs_logfile=True)
passed_safety_005
```

```
## True
```

epsilon = 0.01

```
passed_safety_001, solution_001 = SA_001.run(write_cs_logfile=True)
passed_safety_001
```

```
## True
```

Accuracy

Obtain and store accuracy as an object.

```
# separate the predictor variables from the sensitive variable and the response variable
X_outdf = outdf.drop(columns = ['is_recid', 'Black', 'White'])
X_sens = outdf[['Black', 'White']]
y_outdf = outdf['is_recid']

#----------------------------------#
# get the solution & store coefficients (epsilon = 0.2)
coefficients = SA_02.cs_result["candidate_solution"]

# get the intercept
intercept = coefficients[0]

# compute the predictive values
linear_combination = np.dot(X_outdf, coefficients[1:]) + intercept
pred_probs_02 = 1 / (1 + np.exp(-linear_combination))

#----------------------------------#
# get the solution & store coefficients (epsilon = 0.1)
coefficients = SA_01.cs_result["candidate_solution"]

# get the intercept
intercept = coefficients[0]

# compute the predictive values
linear_combination = np.dot(X_outdf, coefficients[1:]) + intercept
pred_probs_01 = 1 / (1 + np.exp(-linear_combination))
```

```
#----------------------------------#
# get the solution & store coefficients (epsilon = 0.05)
coefficients = SA_005.cs_result["candidate_solution"]

# get the intercept
intercept = coefficients[0]

# compute the predictive values
linear_combination = np.dot(X_outdf, coefficients[1:]) + intercept
pred_probs_005 = 1 / (1 + np.exp(-linear_combination))

#----------------------------------#
# get the solution & store coefficients (epsilon = 0.01)
coefficients = SA_001.cs_result["candidate_solution"]

# get the intercept
intercept = coefficients[0]

# compute the predictive values
linear_combination = np.dot(X_outdf, coefficients[1:]) + intercept
pred_probs_001 = 1 / (1 + np.exp(-linear_combination))

# store results
seldonian_results = pd.DataFrame({'is_recid': y_outdf, 'pred_0.2': pred_probs_02, 'pred_0.1': pred_probs
seldonian_results = pd.concat([X_outdf, X_sens, seldonian_results], axis = 1)

# define threshold
threshold = 0.5

# create risk columns
risk_02 = np.where(pred_probs_02 >= threshold, 1, 0)
risk_01 = np.where(pred_probs_01 >= threshold, 1, 0)
risk_005 = np.where(pred_probs_005 >= threshold, 1, 0)
risk_001 = np.where(pred_probs_001 >= threshold, 1, 0)

# add risk columns to dataframe
seldonian_results['risk_0.2'] = risk_02
seldonian_results['risk_0.1'] = risk_01
seldonian_results['risk_0.05'] = risk_005
seldonian_results['risk_0.01'] = risk_001

# write the dataframe to a CSV file
seldonian_results.to_csv("/home/dasienga24/Statistics-Senior-Honors-Thesis/Data Sets/COMPAS/Simulation S
```

```
# read in the data
seldonian_results <- read.csv("/home/dasienga24/Statistics-Senior-Honors-Thesis/Data Sets/COMPAS/Simulat
```

epsilon = 0.2

```
sa_0.2_accuracy <- count(seldonian_results$risk_0.2 == seldonian_results$is_recid)/nrow(seldonian_result
sa_0.2_accuracy
```

```
## n_TRUE
```

```
## 0.7616
```

epsilon = 0.1

```
sa_0.1_accuracy <- count(seldonian_results$risk_0.1 == seldonian_results$is_recid)/nrow(seldonian_resul
sa_0.1_accuracy
```

```
## n_TRUE
## 0.6832
```

epsilon = 0.05

```
sa_0.05_accuracy <- count(seldonian_results$risk_0.05 == seldonian_results$is_recid)/nrow(seldonian_resu
sa_0.05_accuracy
```

```
## n_TRUE
##  0.595
```

epsilon = 0.01

```
sa_0.01_accuracy <- count(seldonian_results$risk_0.01 == seldonian_results$is_recid)/nrow(seldonian_resu
sa_0.01_accuracy
```

```
## n_TRUE
##    0.5
```

Discrimination

Obtain and store the discrimination statistic as an object.

```
seldonian_results <- seldonian_results %>%
  mutate(race = ifelse(Black == 1, 'Black', 'White'),
         pred_risk_0.2 = ifelse(risk_0.2 == 0, 'Low', 'High'),
         pred_risk_0.1 = ifelse(risk_0.1 == 0, 'Low', 'High'),
         pred_risk_0.05 = ifelse(risk_0.05 == 0, 'Low', 'High'),
         pred_risk_0.01 = ifelse(risk_0.01 == 0, 'Low', 'High'))
```

epsilon = 0.2

```
discrimination <- seldonian_results %>%
  dplyr::select(race, pred_risk_0.2, is_recid) %>%
  group_by(race, is_recid) %>%
  mutate(total = n()) %>%
  group_by(pred_risk_0.2, race, total) %>%
  summarise("reoffended" = count(is_recid == 1),
            "did_not_reoffend" = count(is_recid == 0)) %>%
  pivot_longer(cols = c("reoffended", "did_not_reoffend"),
               names_to = "recidivism") %>%
  pivot_wider(
    id_cols = c("pred_risk_0.2", "recidivism", "total"),
    names_from = "race",
    values_from = value
  ) %>%
  mutate(Black = round(100 * Black / total, 2),
```

```
        White = round(100 * White / total, 2)) %>%
  dplyr::select(-total) %>%
  group_by(pred_risk_0.2, recidivism) %>%
  summarize(Black = max(Black, na.rm = TRUE),
            White = max(White, na.rm = TRUE)) %>%
  filter((pred_risk_0.2 == "High" & recidivism == "did_not_reoffend") |
           (pred_risk_0.2 == "Low" & recidivism == "reoffended")
  )

sa_0.2_disc_stat <- sum(abs(discrimination$White - discrimination$Black))/100
sa_0.2_disc_stat
```

```
## [1] 0.1696
```

epsilon = 0.1

```
discrimination <- seldonian_results %>%
  dplyr::select(race, pred_risk_0.1, is_recid) %>%
  group_by(race, is_recid) %>%
  mutate(total = n()) %>%
  group_by(pred_risk_0.1, race, total) %>%
  summarise("reoffended" = count(is_recid == 1),
            "did_not_reoffend" = count(is_recid == 0)) %>%
  pivot_longer(cols = c("reoffended", "did_not_reoffend"),
               names_to = "recidivism") %>%
  pivot_wider(
    id_cols = c("pred_risk_0.1", "recidivism", "total"),
    names_from = "race",
    values_from = value
  ) %>%
  mutate(Black = round(100 * Black / total, 2),
         White = round(100 * White / total, 2)) %>%
  dplyr::select(-total) %>%
  group_by(pred_risk_0.1, recidivism) %>%
  summarize(Black = max(Black, na.rm = TRUE),
            White = max(White, na.rm = TRUE)) %>%
  filter((pred_risk_0.1 == "High" & recidivism == "did_not_reoffend") |
           (pred_risk_0.1 == "Low" & recidivism == "reoffended")
  )

sa_0.1_disc_stat <- sum(abs(discrimination$White - discrimination$Black))/100
sa_0.1_disc_stat
```

```
## [1] 0.0624
```

epsilon = 0.05

```
discrimination <- seldonian_results %>%
  dplyr::select(race, pred_risk_0.05, is_recid) %>%
  group_by(race, is_recid) %>%
  mutate(total = n()) %>%
  group_by(pred_risk_0.05, race, total) %>%
  summarise("reoffended" = count(is_recid == 1),
            "did_not_reoffend" = count(is_recid == 0)) %>%
```

```r
  pivot_longer(cols = c("reoffended", "did_not_reoffend"),
               names_to = "recidivism") %>%
  pivot_wider(
    id_cols = c("pred_risk_0.05", "recidivism", "total"),
    names_from = "race",
    values_from = value
  ) %>%
  mutate(Black = round(100 * Black / total, 2),
         White = round(100 * White / total, 2)) %>%
  dplyr::select(-total) %>%
  group_by(pred_risk_0.05, recidivism) %>%
  summarize(Black = max(Black, na.rm = TRUE),
            White = max(White, na.rm = TRUE)) %>%
  filter((pred_risk_0.05 == "High" & recidivism == "did_not_reoffend") |
           (pred_risk_0.05 == "Low" & recidivism == "reoffended")
  )

sa_0.05_disc_stat <- sum(abs(discrimination$White - discrimination$Black))/100
sa_0.05_disc_stat
```

```
## [1] 0.0328
```

epsilon = 0.01

```r
discrimination <- seldonian_results %>%
  dplyr::select(race, pred_risk_0.01, is_recid) %>%
  group_by(race, is_recid) %>%
  mutate(total = n()) %>%
  group_by(pred_risk_0.01, race, total) %>%
  summarise("reoffended" = count(is_recid == 1),
            "did_not_reoffend" = count(is_recid == 0)) %>%
  pivot_longer(cols = c("reoffended", "did_not_reoffend"),
               names_to = "recidivism") %>%
  pivot_wider(
    id_cols = c("pred_risk_0.01", "recidivism", "total"),
    names_from = "race",
    values_from = value
  ) %>%
  mutate(Black = round(100 * Black / total, 2),
         White = round(100 * White / total, 2)) %>%
  dplyr::select(-total) %>%
  group_by(pred_risk_0.01, recidivism) %>%
  summarize(Black = max(Black, na.rm = TRUE),
            White = max(White, na.rm = TRUE)) %>%
  filter((pred_risk_0.01 == "High" & recidivism == "did_not_reoffend") |
           (pred_risk_0.01 == "Low" & recidivism == "reoffended")
  )

sa_0.01_disc_stat <- sum(abs(discrimination$White - discrimination$Black))/100
sa_0.01_disc_stat
```

```
## [1] 0
```

Results

Synthesize results for reporting.

```
# create a vector of accuracies
accuracy_vector <- c(lr_accuracy, sa_0.2_accuracy, sa_0.1_accuracy, sa_0.05_accuracy, sa_0.01_accuracy)

# transpose the vector to create a data frame with 5 columns
accuracy_df <- as.data.frame(t(accuracy_vector))

# name the columns of the data frame
colnames(accuracy_df) <- c("LR", "SA (0.2)", "SA (0.1)", "SA (0.05)", "SA (0.01)")

accuracy_df %>%
  kable(caption = "Model Accuracy")
```

Table 1: Model Accuracy

LR	SA (0.2)	SA (0.1)	SA (0.05)	SA (0.01)
0.7796	0.7616	0.6832	0.595	0.5

```
# create a vector of discrimination
disc_vector <- c(lr_disc_stat, sa_0.2_disc_stat, sa_0.1_disc_stat, sa_0.05_disc_stat, sa_0.01_disc_stat)

# transpose the vector to create a data frame with 5 columns
disc_df <- as.data.frame(t(disc_vector))

# name the columns of the data frame
colnames(disc_df) <- c("LR", "SA (0.2)", "SA (0.1)", "SA (0.05)", "SA (0.01)")

disc_df %>%
  kable(caption = "Model Discrimination")
```

Table 2: Model Discrimination

LR	SA (0.2)	SA (0.1)	SA (0.05)	SA (0.01)
0.2464	0.1696	0.0624	0.0328	0

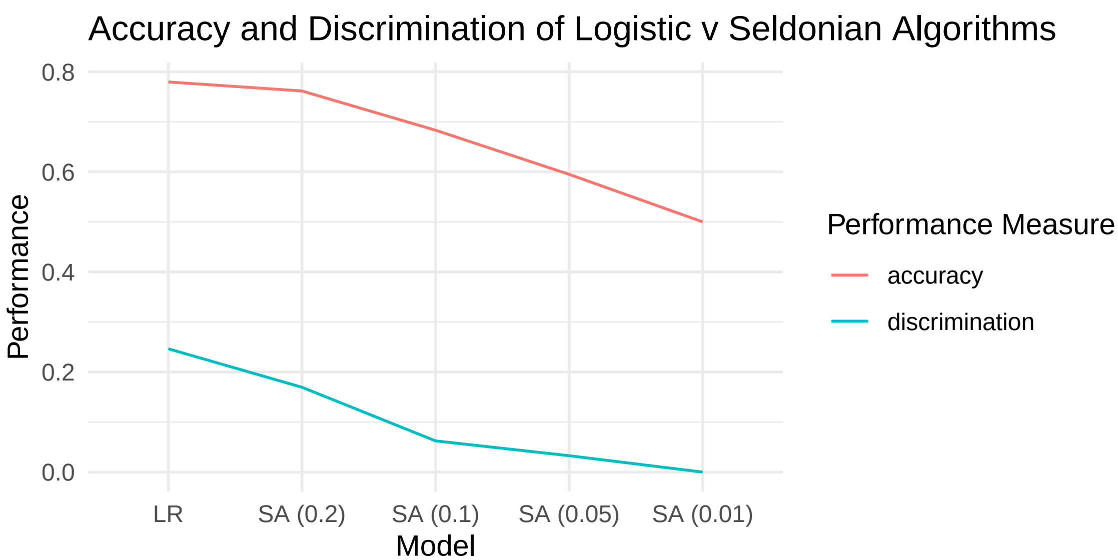
```
plot_data <- accuracy_df %>%
  union(disc_df) %>%
  pivot_longer(cols = c("LR", "SA (0.2)", "SA (0.1)", "SA (0.05)", "SA (0.01)"),
               names_to = "model", values_to = "value") %>%
  mutate(statistic = ifelse(row_number() <= 5, "accuracy", "discrimination"))
```

```
# define the desired order of the x-axis
desired_order <- c("LR", "SA (0.2)", "SA (0.1)", "SA (0.05)", "SA (0.01)")

# convert the 'model' variable to a factor with the desired order
plot_data$model <- factor(plot_data$model, levels = desired_order)

# plot
ggplot(data = plot_data, mapping = aes(x = model,
```

```r
                                              y = value,
                                              color = statistic,
                                              group = statistic)) +
  geom_line() +
  theme_minimal() +
  labs(x = "Model",
       y = "Performance",
       color = "Performance Measure",
       title = "Accuracy and Discrimination of Logistic v Seldonian Algorithms")
```



Note

Before scaling this process, I will need to separately handle and track instances where the Seldonian algorithm does not converge, in case such cases emerge.