Final Project: Predicting Covid Vaccine Side effects

Qinyue Hao

12/21/2021

1 Introduction

I fit three supervised learning models to the reported COVID-19 Vaccine side effect data of 2021 in the United States from the Vaccine Adverse Event Reporting System (VAERS), to predict whether the patient recovered from the side effect. And in the end I found the random forest model with

2 Data set: VAERS 2021 Data Set

The data is a combination of the Covid-19 vaccine side effect data and COVID-19 vaccine manufacture information in 2021 collected by The Vaccine Adverse Event Reporting System (VAERS). The U.S. Department of Health and Human Services (DHHS) established VAERS, which is co-administered by the FDA and the CDC, to accept all reports of suspected adverse events, in all age groups, after the administration of any U.S. licensed vaccine.

Link to whole dataset: https://vaers.hhs.gov/data/datasets.html? Link to the 2021 dataset: https://vaers.hhs.gov/eSubDownload/index.jsp?fn=2021VAERSData.zip

2.1 Variable Description

Here are the brief descriptions of the variables I initially tried to include in my supervised learning models.

2.1.1 Dependent variable

- RECOVD: Whether the vaccinee has recovered from COVID-19 vaccine side effects, can be "Y" for yes, "N" for no, and "U" for unknown.

2.1.2 Independent variables

Numeric

- CAGE_YR: Calculated age of patient in years
- HOSPDAYS: Number of days hospitalized
- NUMDAYS: Number of days (onset date – vaccination date)

Categorical

- STATE: The home state of the vaccinee
- SEX: Sex
- ER_VISIT: Emergency room or doctor visit
- HOSPITAL: Hospitalized
- X_STAY: Prolongation of existing hospitalization
- DISABLE: Disability
- V_ADMINBY: Type of facility where vaccine was administered
- V_FUNDBY: Type of funds used to purchase vaccines
- BIRTH_DEFECT: Congenital anomaly or birth defect
- OFC_VISIT: Doctor or other healthcare provider office/clinic visit
- ER_ED_VISIT: Emergency room/ department or urgent care

- VAX_MANU: Vaccine manufacturer
- VAX_LOT: Manufacturer's vaccine lot
- VAX_DOSE_SERIES: Number of doses administered
- VAX_ROUTE: Vaccination route
- VAX_SITE: Vaccination site

Date

- RECVDATE: Date report was received
- VAX_DATE: Vaccination date

*See more detailed variable descriptions here: https://vaers.hhs.gov/docs/VAERSDataUseGuide_en_September2021.pdf

3 Data Preprocessing

```r
setwd("/Users/hailey/Documents/GitHub/DataMining/project")
d1 <- read_csv("2021VAERSData/2021VAERSDATA.csv")
d2 <- read_csv("2021VAERSData/2021VAERSVAX.csv")
d <- inner_join(d1, d2, by = "VAERS_ID") %>% filter(VAX_TYPE == "COVID19")
```

```
# remove redundant variables or variables not of interest (not interested in digging into the details o
drop_var <- c("VAX_TYPE", "AGE_YRS", "CAGE_MO", "DATEDIED", "VAERS_ID", "DIED", "TODAYS_DATE", "DATEDIE
# text_var <- c("SYMPTOM_TEXT", "L_THREAT", "LAB_DATA", "OTHER_MEDS", "CUR_ILL", "HISTORY", "PRIOR_VAX"
drop_text_var <- c("L_THREAT", "LAB_DATA", "OTHER_MEDS", "CUR_ILL", "HISTORY", "PRIOR_VAX", "ALLERGIES"
d <- d %>%
  select(-all_of(drop_var)) %>%
  select(-all_of(drop_text_var))
head(d)
```

```
missingness <- ff_glimpse(d)
mContinuous <- missingness$Continuous
mCategorical <- missingness$Categorical

drop_missing_categorical <- mCategorical %>% filter(missing_percent >= 50)
drop_missing_continuous <- mContinuous %>% filter(missing_percent >= 50)

drop_missing_categorical$label
drop_missing_continuous$label

# For variables below, NAs are recoded as a category "U", which means "unknown" (to match "Y")
N_var <- c("HOSPITAL", "X_STAY", "DISABLE", "BIRTH_DEFECT", "OFC_VISIT", "ER_ED_VISIT")
# d[N_var]
d[N_var][is.na(d[N_var])] <- 'U'
# d[N_var]
# remove variables with too much missing values (over 50%, my random choice) and not otherwise recoded
d <- d %>%
  select(-c("ONSET_DATE", "V_FUNDBY", "VAX_LOT")) %>%
  select(-HOSPDAYS)

head(d)

d <- d %>%
  mutate(RPTDAYS = as.integer(as.Date(RECVDATE, format =  "%m/%d/%Y") - as.Date(VAX_DATE, format =  "%m
  mutate(OSTDAYS = NUMDAYS) # days between vaccine and side effect onset
d <- select(d, -c("NUMDAYS", "RECVDATE", "VAX_DATE"))

# ff_glimpse(d)
# Since we have a large dataset...
# remove rows with missing value in the outcome variable (impute the rest with knn clustering as a reci
# and recode outcome variable as factor
d0 <- d %>%
  filter(!is.na(RECOVD)) %>%
  filter(RECOVD != "U") %>%
  na.omit
# convert all character variables to factor variable
# d0[sapply(d0, is.character)] <- lapply(d0[sapply(d0, is.character)], as.factor)

write.csv(d0, "clean.csv")
# colnames(d) # 18 variables, including 1 text variable
```

```
# # fill the missing values ("CAGE_YR" and "NUMDAYS")
# library(mice)
# df <- mice(d, verbose = F)
```

```r
# df <- complete(df)
# write.csv(df, "fill_clean.csv")
```

In the data pre-processing,

First, I removed the redundant variables with similar information with some other variable.

Second, I recoded some NAs in some variables to an category "U" indicating "unknown".

Third, I checked the missingness in the data, and dropped a few columns with too much missingness.

Fourth, I removed the "unknown" category in the outcome variable, and proceed with only two categories: "Y" for yes, and "N" for no, indicating whether the vaccinee recovered from the COVID-19 side effect.

Fifth, I removed the rows with missing values.

4 Supervised learning Models

4.1 Train-test split

```r
setwd("/Users/hailey/Documents/GitHub/DataMining/project")
df <- read_csv("clean.csv") %>%
  select(- "...1")

df[sapply(df, is.character)] <- lapply(df[sapply(df, is.character)], as.factor)
df <- df[sample(nrow(df), 200), ]

split <- initial_split(df, prob = 0.8)
train <- training(split)
test <- testing(split)
```

```r
if (.Platform$OS.type == "windows") {
  doParallel::registerDoParallel(parallel::detectCores())
} else doMC::registerDoMC(parallel::detectCores())
```

4.2 Base recipe without text data and bootstrap samples

```r
base_recipe <-
  recipe(RECOVD ~ ., data = train) %>%
  step_rm("SYMPTOM_TEXT") %>% # text data will be used in different ways
  # step_impute_knn(all_predictors()) %>%
  step_dummy(all_nominal_predictors()) %>%
  # step_string2factor((all_nominal_predictors())) %>%
  step_normalize(all_numeric_predictors()) %>%
  prep(training = train)


base_bs <- bootstraps(train, times = 10)
```

4.3 Models

I used a smaller sample, fewer bootstraps, and tuning levels to speed up the knitting process. I used a sample of 3000 observations, bootstrap 50 times, 10 tuning levels for the parameters in each model to get the results I wrote in the conclusions. The results are different every time I ran it, so I just wrote the conclusions based on the ones I got in the console just before knitting, so it may be somewhat different from what's the code output on the pdf.

4.3.1 Penalized Logistic Regression (glmnet)

```
glmnet_model <-
  logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")

glmnet_wf <-
  workflow() %>%
  add_recipe(base_recipe) %>%
  add_model(glmnet_model)

glmnet_bs <- base_bs

glmnet_grid <- grid_regular(parameters(glmnet_model), levels = 3)

results <- tune_grid(glmnet_wf,
                     resamples = glmnet_bs,
                     grid = glmnet_grid,
                     metrics = metric_set(accuracy)) # classification

glmnet_best <- select_best(results, metric = "accuracy")
print(glmnet_best) # penalty = 0.0000000001
```

```
## # A tibble: 1 x 3
##   penalty mixture .config
##     <dbl>   <dbl> <chr>
## 1       1    0.05 Preprocessor1_Model3
```

```
glmnet_final_wf <- finalize_workflow(glmnet_wf, glmnet_best)

glmnet_fit <- fit(glmnet_final_wf, train)

glmnet_pred <- predict(glmnet_fit, new_data = test)

(glmnet_accuracy <- accuracy(glmnet_pred, truth = test$RECOVD, estimate = .pred_class))
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.42
```

```
# 0.707
```

The classification accuracy of the best penalized logistic regression model is 0.707.

4.3.2 Multivariate adaptive regression spline (MARS)

Since the penalized logistic regression did not work very well (with an classification accuracy about 0.7, I tried the Multivariate adaptive regression spline model to see if this more flexible non-linear extension of linear regression models can predict any better.

```
mars_model <-
  mars(num_terms = tune()) %>%
  set_engine("earth") %>%
  set_mode("classification")

mars_wf <-
  workflow() %>%
  add_recipe(base_recipe) %>%
  add_model(mars_model)

mars_bs <- base_bs

mars_grid <- grid_regular(parameters(mars_model), levels = 3)

results <- tune_grid(mars_wf,
                     resamples = mars_bs,
                     grid = mars_grid,
                     metrics = metric_set(accuracy)) # classification

mars_best <- select_best(results, metric = "accuracy")
print(mars_best) # num_terms = 4
```

```
## # A tibble: 1 x 2
##   num_terms .config
##       <int> <chr>
## 1         5 Preprocessor1_Model3
```

```
mars_final_wf <- finalize_workflow(mars_wf, mars_best)

mars_fit <- fit(mars_final_wf, train)

mars_pred <- predict(mars_fit, new_data = test)

(mars_accuracy <- accuracy(mars_pred, truth = test$RECOVD, estimate = .pred_class))
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.42
```

```
# 0.733
```

It turned out that the MARS model predict better than the penalized logistic regression model, which is 0.733.We may need to go more complicated to get higher accuracy. So next I tried to use tree models, specifically random forest model, which is recognized to generally perform better than linear models.

4.3.3 Random forest

```
rf_model <-
  rand_forest(min_n = tune()) %>% # minimum number of data points in a node that are required for the n
  set_engine("randomForest",
             num.threads = parallel::detectCores(),
             importance = TRUE,
             verbose = TRUE) %>%
  set_mode("classification")

rf_wf <- workflow() %>%
  add_model(rf_model) %>%
  add_recipe(base_recipe)

rf_bs <- base_bs

rf_grid <- grid_regular(parameters(rf_model), levels = 3)
# rf_grid <- tibble(tree = c(200, 400, 600, 800, 1000))

results <- tune_grid(rf_wf,
                      resamples = rf_bs,
                      grid = rf_grid,
                      metrics = metric_set(accuracy)) # classification

rf_best <- select_best(results, metric = "accuracy")
print(rf_best) # min_n = 23
```

```
## # A tibble: 1 x 2
##   min_n .config
##   <int> <chr>
## 1    40 Preprocessor1_Model3
```

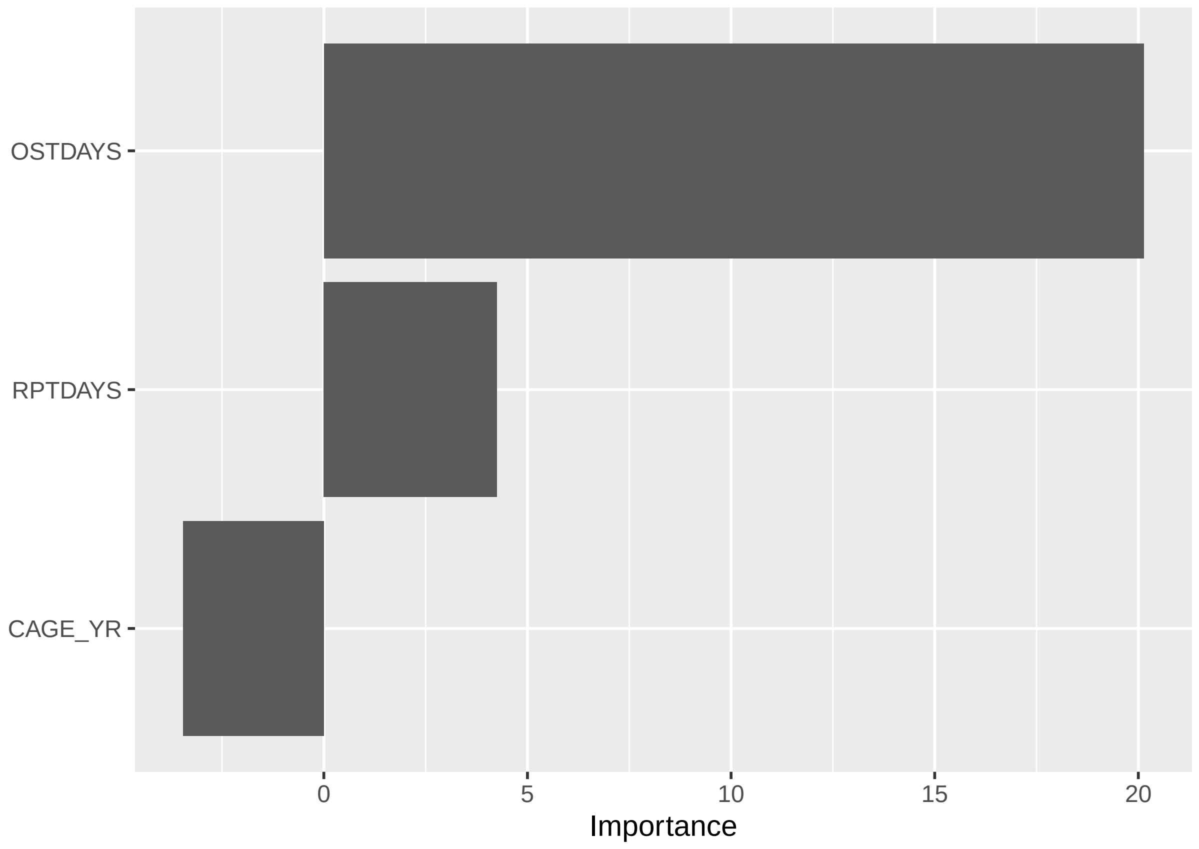
```
rf_final_wf <- finalize_workflow(rf_wf, rf_best)

rf_fit <- fit(rf_final_wf, train)
# rf_fit <- fit(rf_wf, train)

# bind_cols(test,
#            predict(rf_fit, new_data = test)) %>%
#   accuracy(truth = RECOVD, estimate = .pred_class)
rf_pred <- predict(rf_fit, new_data = test)
(rf_accuracy <- accuracy(rf_pred, truth = test$RECOVD, estimate = .pred_class))
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.58
```

```
# 0.747
```

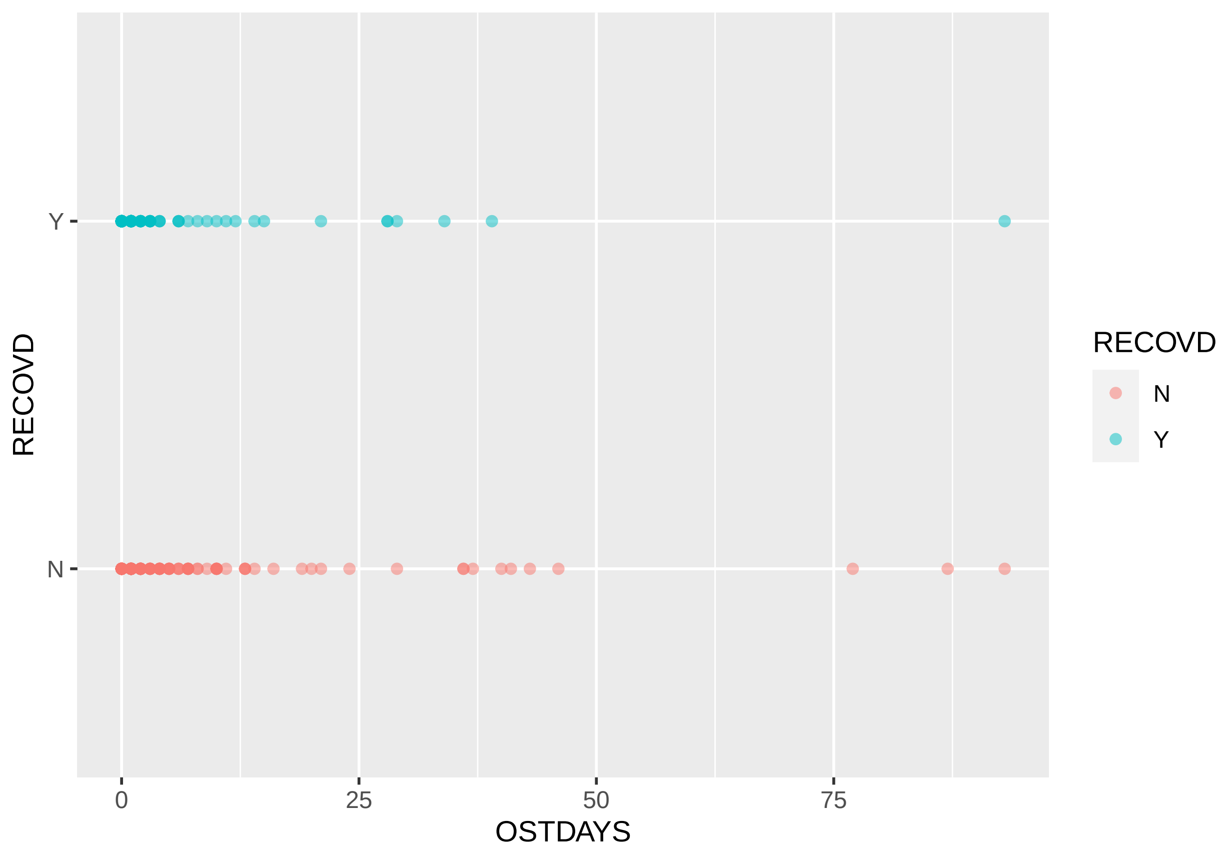
As expected, the random forest model performs notably better than the former two models, it's classification accuracy on the testing data is 0.747.

```
extract_fit_parsnip(rf_fit) %>% vip
```



The graph above shows that splitting side effect onset lag into smaller and smaller categories helped increasing the predicting accuracy more than the splitting on the lag between vaccination and reporting. Splitting on age did little help in this case. As shown below, by plotting the relationship between the recovery state and side effect onset lag, it seems there's some difference between the two categories that can be studies further afterwards – those who recovered seems mostly recovered in 10 days.

```
library(ggplot2)
df %>% filter(OSTDAYS <= 100) %>%
  ggplot(aes(OSTDAYS, RECOVD)) +
  geom_point(aes(color = RECOVD), alpha = 0.5)
```



4.3.4 Single layer neural network

```r
nnet_model <-
  mlp(epochs = 100,
      hidden_units = 32,
      penalty = tune(),
      activation = "relu") %>%
  set_mode("classification") %>%
  set_engine("nnet", verbose = 0)

nnet_wf <-
  workflow() %>%
  add_recipe(base_recipe) %>%
  add_model(nnet_model)

nnet_bs <- base_bs

nnet_grid <- grid_regular(parameters(nnet_model), levels = 3)

results <- tune_grid(nnet_wf,
                     resamples = nnet_bs,
                     grid = nnet_grid,
                     metrics = metric_set(accuracy)) # classification
```

```r
nnet_best <- select_best(results, metric = "accuracy")
print(nnet_best) # penalty = 0.00599
```

```
## # A tibble: 1 x 2
##   penalty .config
##     <dbl> <chr>
## 1 0.00001 Preprocessor1_Model2
```

```r
nnet_final_wf <- finalize_workflow(nnet_wf, nnet_best)

nnet_fit <- fit(nnet_final_wf, data = train)
# nnet_fit <- fit(nnet_wf, data = train)

nnet_pred <- predict(nnet_fit, new_data = test)
(nnet_accuracy <- accuracy(nnet_pred, truth = test$RECOVD, estimate = .pred_class) )
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.62
```

The Single layer neural network did not predict very very, accuracy score on test set is only 0.667.

5 Conclusions

Among the four types of supervised models I tried above, Penalized Logistic Regression Model, Multivariate Adaptive Regression Spline, Random Forest Model, Penalized Single layer Neural Network Model, as representatives of linear models, regression splines models, tree models and neural networks, **the random forest model** (min_n = 23)performed the best with our data. In the end, with the variables I included in the model, my best model, the random forest model could predict the recovery state of the vaccinee with around 75% accuracy.

6 Future directions

By using only the eligible numeric and categorical variables in this data set, I was not able to predict the recovery state of a vaccinee reported COVID-19 vaccine side effect very well, only end up with a best classification accuracy of 75% or so. There is some important information not used here describing the symptoms the vaccinee had, as well as past medical record, allergies and so on. I'll try to improve the prediction accuracy by utilizing the information in those text variables, with a combination of supervised learning and NLP methods.