Final Project

Vasiliy Ostapenko (774 970 8)

May 30, 2022

Contents

INTRO

The purpose of this project is to generate a model that will predict whether a person recovered from their vaccine side effects given features about the person, the vaccine, and their symptoms.

What is VAERS?

The Vaccine Adverse Effect Reporting System (VAERS) database is a joint effort by the FDA and the CDC to provide a system for reporting both minor and serious events related to vaccines. In this project, I will be using the VAERS dataset for calendar year 2021. One could find further information as well as the data at the following link: https://vaers.hhs.gov/data.html. The data is split into three tables, called "data", "symptoms", and "vaccine", respectively. Each row of every table is a separate event, with a unique ID attached. The unique event ID could be used to map between the three tables. The "data" table also provides information on the patient, their symptoms, and their treatment. The "symptoms" table additionally lists symptoms related to each event in further detail and codes them in the internationally-accepted medDRA format. Finally, the "vaccine" table gives further information on the vaccine related to each event.

Prior preprocessing

Before building the R predictive models, I leveraged Python to preprocess and clean the data. This choice was made due to the language's capability and the fact that some useful functions (implemented in various packages) are not easily replicated in R. In two Jupyter notebooks (process_pt1.ipynb, process_pt2.ipynb), I read in the three tables, one at a time, and transformed the data for use in a machine learning context. I remapped categorical data to numeric equivalents, filtered none-type values, created some derived columns to remove redundant features, and so on. Finally, after negotiating duplicate rows, I merged the three tables into one frame (combined.csv), which is used here.

Why might this model be useful?

A healthcare provider might encounter a patient presenting with vaccine side effects. They might note the patient profile, the vaccine type, as well as some of the symptoms. Using this model, they might be able to predict whether the person will recover from their ailments.

DATA

Load Data

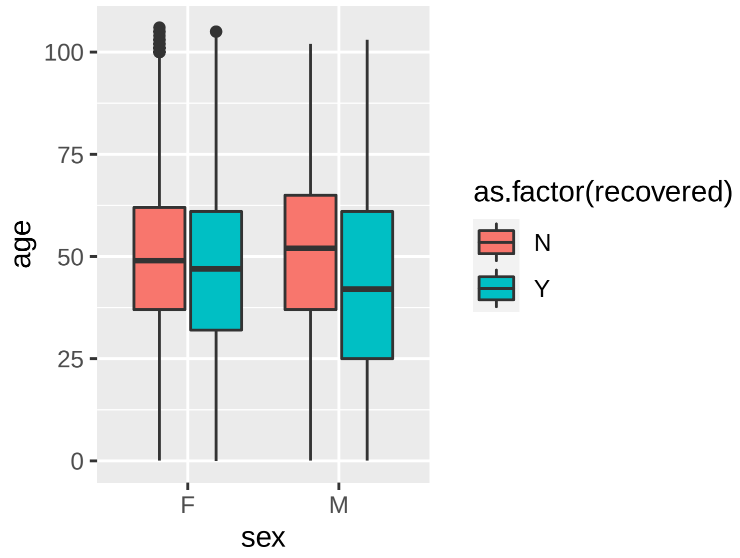
- Dataset variables (included in codebook):
  - vaersId: unique identifier for an instance of an adverse effect
  - age: patient age
  - sex: patient sex
  - recovered: indicator, whether patient recovered
  - deltaOnset: time between vaccination and symptom onset
  - adminBy: dummy-coded, type of healthcare provider who administered vaccine
  - otherMeds: dummy-coded, type of medications the patient is on
  - history: dummy-coded, type of prior patient history
  - region: dummy-coded, US region to which the patient's state belongs
  - deltaReceived: time between symptom onset and receipt of VAERS report
  - nCovidVax: number of COVID vaccine doses the patient received
  - s1-s5: dummy-coded, first five symptoms the patient experienced
  - myocarditis: indicator, whether patient experienced myocarditis

```
# Read in data
DATA_FOLDER = "./data/processed/"
COMBINED_FNAME = file.path(DATA_FOLDER, "combined.csv")
df = read.csv(COMBINED_FNAME) %>%
  column_to_rownames("vaersId")
```

Visualization

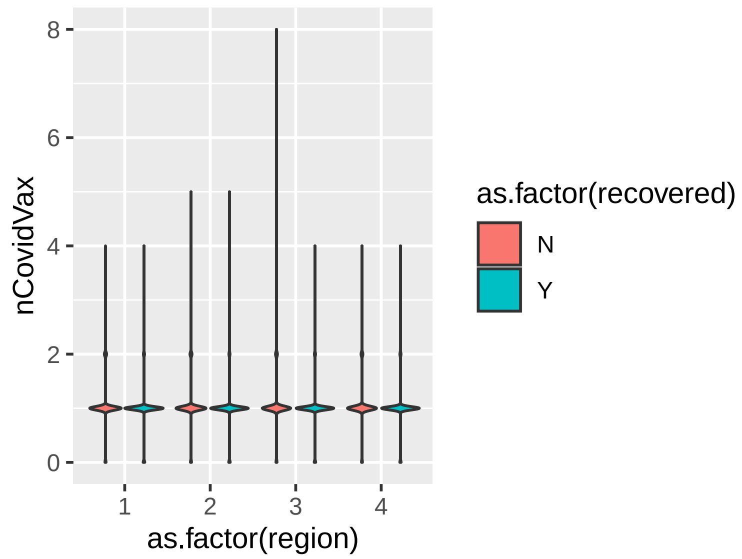
The first plot is a boxplot of sex by age, colored by recovered. I believe that younger people will be more likely to recover from their symptoms.

```
# Plot 1
g1 = ggplot(df, aes(x=sex, y=age, fill=as.factor(recovered))) +
  geom_boxplot()
g1
```



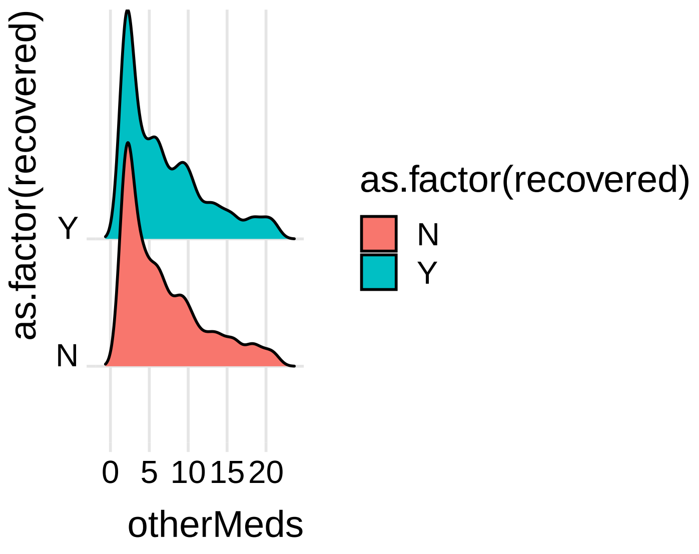
The second plot is a violin plot of region by nCovidVax, colored by recovered. I believe that there will be regional differences in vaccination rates. I also think that there might be a relationship between number of covid vaccine doses and ability to recover.

```r
# Plot 2
g2 = ggplot(df, aes(x=as.factor(region), y=nCovidVax, fill=as.factor(recovered))) +
  geom_violin()
g2
```



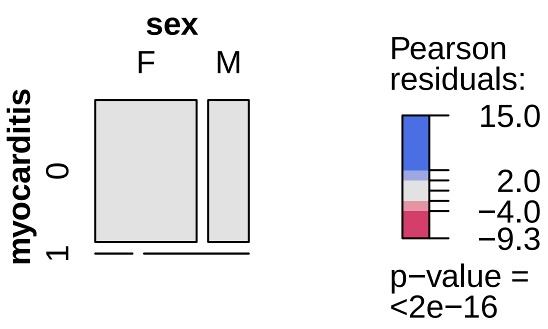
The third plot is a ridge plot of otherMeds, split by recovered. I believe that there might be a difference in the types of medications people who recover and people who don't are on. Perhaps younger, healthier individuals who recover will tend to not need the same medications as older, sicker people.

```r
# Plot 3
g3 = ggplot(df[!(df$otherMeds %in% c(0, 1)), ], aes(x=otherMeds, y=as.factor(recovered), fill=as.factor
  geom_density_ridges() +
  theme_ridges()
g3
```



The fourth, and final, plot is a mosaic plot of myocarditis by sex. I believe that there is a relationship between vaccine induced myocarditis and sex.

```
# Plot 4
p4 = mosaic(~myocarditis+sex, data=df, shade=TRUE, legend=TRUE)
```



```
p4
```

```
##              sex      F      M
## myocarditis
## 0                199530  80009
## 1                    80    225
```

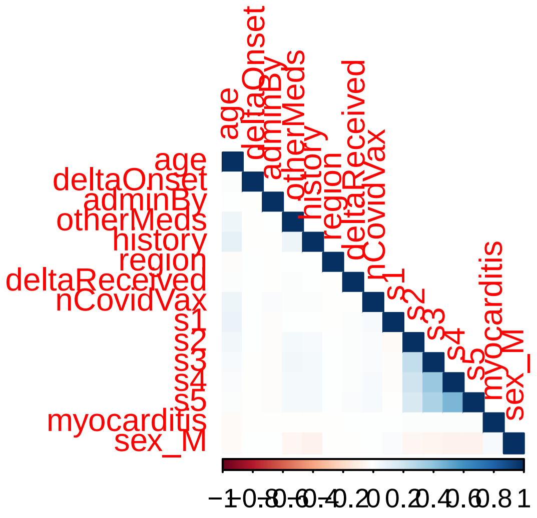
Categorical to Numeric Conversion

Before we get started with the data splitting and model building, I will convert the two remaining categorical variables, sex and recovered, into numeric dummies.

```
df = fastDummies::dummy_cols(df, remove_first_dummy=TRUE, remove_selected_columns=TRUE)
```

I will also inspect the column by column correlation to remove redundant features which are correlated with some other columns.

```
corrplot(cor(df[ , names(df) != "recovered_Y"]),
         method="color", type="lower")
```



From the plot, we glean that the myocarditis column is highly correlated with s4 and s5 columns. Thus it will be removed.

```
df = df[ , !(colnames(df) %in% c("myocarditis"))] %>% copy()
df$recovered_Y = as.factor(df$recovered_Y)
```

Data Split

I will split the data using a 70/30 train and test split. I will also use 3-fold cross validation when I am fitting my models and tuning hyperparameters.

```
split = df %>%
  initial_split(prop=0.70, strata="recovered_Y")

train = training(split)
test = testing(split)
folds = vfold_cv(train, v=3, strata="recovered_Y")
```

MODELING

Recipe

Due to the extensive prior preprocessing, the only intermediate step in the recipe will be to normalize all predictors. Normalization enables gradient descent to converge faster and to the correct minimum.

```
rec = recipe(recovered_Y ~ ., data=train) %>%
  step_normalize(all_predictors())
```

Models, Workflows, Parameters, CV

For the classification task, I will fit the following models to the training data: logistic regression; SVM poly kernel; random forest; boosted trees.

For the logistic regression model, I will tune the penalty and mixture hyperparameters, at 5 levels each for a total of 75 models fit.

```
# Logistic Regression
mod_glm = logistic_reg(penalty=tune(), mixture=tune()) %>%
  set_engine("glm") %>%
  set_mode("classification")

work_glm = workflow() %>%
  add_model(mod_glm) %>%
  add_recipe(rec)

grid_glm = grid_regular(penalty(), mixture(), levels=5)

tune_glm = work_glm %>%
  tune_grid(resamples=folds, grid=grid_glm,
            metrics=metric_set(roc_auc, accuracy))

save(tune_glm, work_glm, file="./data/models/tune_glm.rda")
```

```
load(file="./data/models/tune_glm.rda")
tune_glm %>% collect_metrics() %>%
  select(-.estimator, -.config)
```

```
## # A tibble: 2 x 4
##   .metric   mean     n  std_err
##   <chr>    <dbl> <int>    <dbl>
## 1 accuracy 0.563     3 0.000397
## 2 roc_auc  0.600     3 0.000743
```

For the SVM model, I will tune the cost and degree hyperparameters, at 2 levels each for a total of 12 models fit.

```
# SVM
mod_svm = svm_poly(cost=tune(), degree=tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

work_svm = workflow() %>%
  add_model(mod_svm) %>%
  add_recipe(rec)

grid_svm = grid_regular(cost(), degree(), levels=2)

tune_svm = work_svm %>%
  tune_grid(resamples=folds, grid=grid_svm,
            metrics=metric_set(roc_auc, accuracy))

save(tune_svm, work_svm, file="./data/models/tune_svm.rda")
```

```
load(file="./data/models/tune_svm.rda")
tune_svm %>% collect_metrics() %>%
  select(-.estimator, -.config)
```

For the RF model, I will tune the min_n and mtry hyperparameters, at 2 levels each, for a total of 12 models fit.

```
# Random Forest
mod_rf = rand_forest(min_n=tune(), mtry=tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification")
```

```
work_rf = workflow() %>%
  add_model(mod_rf) %>%
  add_recipe(rec)

grid_rf = grid_regular(min_n(), mtry(range=c(1, 14)), levels=2)

tune_rf = work_rf %>%
  tune_grid(resamples=folds, grid=grid_rf,
            metrics=metric_set(roc_auc, accuracy))

save(tune_rf, work_rf, file="./data/models/tune_rf.rda")
```

```
load(file="./data/models/tune_rf.rda")
tune_rf %>% collect_metrics() %>%
  select(-.estimator, -.config)
```

```
## # A tibble: 8 x 6
##    mtry min_n .metric   mean     n std_err
##   <int> <int> <chr>    <dbl> <int>   <dbl>
## 1     1     2 accuracy 0.688     3 0.00112
## 2     1     2 roc_auc  0.755     3 0.00152
## 3     1    40 accuracy 0.687     3 0.00107
## 4     1    40 roc_auc  0.754     3 0.00158
## 5    14     2 accuracy 0.715     3 0.00148
## 6    14     2 roc_auc  0.791     3 0.00120
## 7    14    40 accuracy 0.719     3 0.00179
## 8    14    40 roc_auc  0.795     3 0.00151
```

Finally, for the boosted trees model, I will tune min_n, learn_rate, and mtry. 3 levels each means we will fit a total of 81 models.

```
# Boosted Trees
mod_boost = boost_tree(min_n=tune(), learn_rate=tune(), mtry=tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

work_boost = workflow() %>%
  add_model(mod_boost) %>%
  add_recipe(rec)

grid_boost = grid_regular(min_n(), learn_rate(), mtry(range=c(1, 14)), levels=3)

tune_boost = work_boost %>%
  tune_grid(resamples=folds, grid=grid_boost,
            metrics=metric_set(roc_auc, accuracy))

save(tune_boost, work_boost, file="./data/models/tune_boost.rda")
```

```
load(file="./data/models/tune_boost.rda")
tune_boost %>% collect_metrics() %>%
  select(-.estimator, -.config)
```

```
## # A tibble: 54 x 7
##     mtry min_n   learn_rate .metric   mean     n    std_err
##    <int> <int>        <dbl> <chr>    <dbl> <int>      <dbl>
##  1     1     2 0.0000000001 accuracy 0.493     3 0.00000251
##  2     1     2 0.0000000001 roc_auc  0.5       3 0
##  3     1    21 0.0000000001 accuracy 0.493     3 0.00000251
##  4     1    21 0.0000000001 roc_auc  0.5       3 0
##  5     1    40 0.0000000001 accuracy 0.493     3 0.00000251
##  6     1    40 0.0000000001 roc_auc  0.5       3 0
##  7     1     2 0.00000316   accuracy 0.646     3 0.00537
##  8     1     2 0.00000316   roc_auc  0.709     3 0.00564
##  9     1    21 0.00000316   accuracy 0.655     3 0.00716
## 10     1    21 0.00000316   roc_auc  0.717     3 0.00945
## # ... with 44 more rows
```

Best Model Determination and Training

Looking at each models accuracy and AUC ROC metrics, we determine that the best performance was achieved by the random forest class of models. Thus, we will pick the best random forest model, fit it to the entire training set, and save it.

```
tune_best = tune_rf %>%
  select_best("roc_auc")

work_final = work_rf %>%
  finalize_workflow(tune_best)

fit_best = work_final %>%
  fit(train)

save(fit_best, file="./data/models/fit_best.rda")
```

EVALUATION

Best Model Testing and Evaluation

Finally, after training the RF model, we can fit it to the test set and evaluate some performance metrics.

```
load(file="./data/models/fit_best.rda")
```

```
predict_best = augment(fit_best, test)
```

```
roc_auc(data=predict_best, truth=recovered_Y,
        estimate=.pred_1, event_level="second")
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.724
```

```
accuracy(data=predict_best, truth=recovered_Y,
         estimate=.pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.658
```

```
conf_mtx = conf_mat(data=predict_best, truth=recovered_Y,
                    estimate=.pred_class)
conf_mtx$table
```

```
##            Truth
## Prediction     0     1
##          0 30921 18322
##          1 10430 24281
```

CONCLUSION

The VAERS dataset was very interesting due to its novelty for me and gave me a good challenge in working with complicated data as well as the tidymodels workflow. Predicting recovery turned out to be a more difficult task than I had thought, and further efforts are required to build a high-performing model. Perhaps there are better ways to treat patient medication, medical history, and symptom information. Still, I was not surprised that the tree-based methods performed better than others, given their famed status.