Assignment 2 - Language Development in ASD - Making predictions

The Eyes of Kasparov

September 23, 2020

Welcome to the second exciting part of the Language Development in ASD exercise

In this exercise we will delve more in depth with different practices of model comparison and model selection, by first evaluating your models from last time against some new data. Does the model generalize well? Then we will learn to do better by cross-validating models and systematically compare them.

The questions to be answered (in a separate document) are: 1- Discuss the differences in performance of your model in training and testing data 2- Which individual differences should be included in a model that maximizes your ability to explain/predict new data? 3- Predict a new kid's performance (let's call him Bernie) and discuss it against expected performance of the two groups

Learning objectives

- Critically appraise the predictive framework (contrasted to the explanatory framework)
- Learn the basics of machine learning workflows: training/testing, cross-validation, feature selections

Let's go

N.B. There are several datasets for this exercise, so pay attention to which one you are using!

1. The (training) dataset from last time (the awesome one you produced :-)).
2. The (test) datasets on which you can test the models from last time:

- Demographic and clinical data: https://www.dropbox.com/s/5pc05mh5jwvdfjk/demo_test.csv?dl=0
- Utterance Length data: https://www.dropbox.com/s/eegu8fea2entdqv/LU_test.csv?dl=0
- Word data: https://www.dropbox.com/s/cf4p84mzn2p1bev/token_test.csv?dl=0

Exercise 1) Testing model performance

How did your models from last time perform? In this exercise you have to compare the results on the training data and on the test data. Report both of them. Compare them. Discuss why they are different.

- recreate the models you chose last time (just write the model code again and apply it to your training data (from the first assignment))
- calculate performance of the model on the training data: root mean square error is a good measure. (Tip: google the function rmse())
- create the test dataset (apply the code from assignment 1 to clean up the 3 test datasets)
- test the performance of the models on the test data (Tips: google the functions "predict()")
- optional: predictions are never certain, can you identify the uncertainty of the predictions? (e.g. google predictinterval())

```
pacman::p_load(tidyverse,lmerTest, Metrics,caret, tidymodels)

nicify_predictions <- function(predictions) {
  predictions %>%
    as.list %>%
    as_tibble %>%
    gather(key="row_num", value="prediction") %>%
    mutate(row_num = as.integer(row_num))
}

join_preds <- function(df, preds) {
  df %>%
    mutate(row_num = row_number()) %>%
    left_join(preds) %>%
    drop_na()
}

normalize <- function(x) {(x-mean(x, na.rm=T)) / sd(x, na.rm=T)}

CleanUpData <- function(Demo,LU,Word){

  Speech <- merge(LU, Word) %>%
    rename(
      Child.ID = SUBJ,
      Visit=VISIT) %>%
    mutate(
      Visit = as.numeric(str_extract(Visit, "\\d")),
      Child.ID = gsub("\\.","", Child.ID)
      ) %>%
    dplyr::select(
      Child.ID, Visit, MOT_MLU, CHI_MLU, types_MOT, types_CHI, tokens_MOT,
tokens_CHI
    )

  Demo <- Demo %>%
    dplyr::select(
      Child.ID, Visit, Ethnicity, Diagnosis, Gender, Age, ADOS, MullenRaw,
ExpressiveLangRaw, Socialization
```

```r
    ) %>%
    mutate(
      Child.ID = gsub("\\.","", Child.ID)
    )

  Data=merge(Demo,Speech,all=T)

  Data1= Data %>%
     subset(Visit=="1") %>%
     dplyr::select(Child.ID, ADOS, ExpressiveLangRaw, MullenRaw, Socialization)
%>%
     rename(Ados1 = ADOS,
            verbalIQ1 = ExpressiveLangRaw,
            nonVerbalIQ1 = MullenRaw,
            Socialization1 = Socialization)

  Data=merge(Data, Data1, all=T) %>%
    mutate(
      # Child.ID = as.numeric(as.factor(as.character(Child.ID))),
      Visit = as.numeric(as.character(Visit)),
      Gender = recode(Gender,
         "1" = "M",
         "2" = "F"),
      Diagnosis = recode(Diagnosis,
         "A"  = "ASD",
         "B"  = "TD")
    ) %>%
      mutate(Ethnicity = if_else(str_detect(tolower(Ethnicity), "white"),
                              "White", "Other"),
         Child.ID = as_factor(Child.ID))

  return(Data)
}
set.seed(123)
```

Loading data and predicting old models

```r
pacman::p_load(readr,dplyr,stringr,lmerTest,Metrics,caret, tidymodels)

## Clean up function, included to inspire you



# Load training Data
df_train <- read_csv("data_clean.csv") %>%
  mutate(Ethnicity = if_else(str_detect(tolower(Ethnicity), "white"),
                              "White", "Other"),
         Child.ID = as_factor(Child.ID)) %>%
  # Rescale data
  mutate(across(where(is.numeric), normalize))

## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Ethnicity = col_character(),
##   Diagnosis = col_character(),
##   Gender = col_character()
## )

## See spec(...) for full column specifications.

#- recreate the models you chose last time (just write the code again and apply it
to Train Data)
besty_model <- lmerTest::lmer(CHI_MLU ~ Visit + Ethnicity + Diagnosis + Gender +
Age + Socialization +
    MOT_MLU + types_MOT + types_CHI +
    tokens_MOT + tokens_CHI + (1 | Child.ID) + Visit:Diagnosis +
    Visit:Gender + Visit:Socialization + Ethnicity:Age + Ethnicity:types_MOT +
    Ethnicity:types_CHI + Ethnicity:tokens_MOT + Diagnosis:Socialization +
Gender:Age +
    Gender:Socialization + Gender:types_CHI + Gender:tokens_CHI +
    Age:Socialization + Age:MOT_MLU  +
    Socialization:tokens_MOT + MOT_MLU:tokens_MOT + MOT_MLU:tokens_CHI +
    types_MOT:types_CHI + types_CHI:tokens_MOT + tokens_MOT:tokens_CHI,
data=df_train)


#- calculate performance of the model on the training data: root mean square error
is a good measure. (Tip: google the function rmse())

predictions <- predict(besty_model) %>%
  nicify_predictions()

# Adding predictions to data
df_eval <- df_train %>%
  join_preds(predictions)

## Joining, by = "row_num"

# Getting rmse
rmse_model_train <- Metrics::rmse(df_eval$CHI_MLU, df_eval$prediction)

#- create the test dataset (apply the code from assignment 1 or my function to
clean up the 3 test datasets)
# Test data
lu_test <- read_csv("LU_test.csv")
```

```
## Parsed with column specification:
## cols(
##   SUBJ = col_character(),
##   VISIT = col_character(),
##   MOT_MLU = col_double(),
##   MOT_LUstd = col_double(),
##   MOT_LU_q1 = col_double(),
##   MOT_LU_q2 = col_double(),
##   MOT_LU_q3 = col_double(),
##   CHI_MLU = col_double(),
##   CHI_LUstd = col_double(),
##   CHI_LU_q1 = col_double(),
##   CHI_LU_q2 = col_double(),
##   CHI_LU_q3 = col_double()
## )

demo_test <- read_csv("demo_test.csv")

## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Child.ID = col_character(),
##   Ethnicity = col_character(),
##   Diagnosis = col_character(),
##   Birthdate = col_character()
## )
## See spec(...) for full column specifications.

token_test <- read_csv("token_test.csv")

## Parsed with column specification:
## cols(
##   SUBJ = col_character(),
##   VISIT = col_character(),
##   types_MOT = col_double(),
##   types_CHI = col_double(),
##   types_shared = col_double(),
##   tokens_MOT = col_double(),
##   tokens_CHI = col_double(),
##   X = col_logical()
## )

data_test <- CleanUpData(demo_test, lu_test, token_test) %>%
  mutate(across(where(is.numeric),  normalize)) %>%
  mutate(row_num = row_number())



#- test the performance of the models on the test data (Tips: google the functions
"predict()")

# calculating predictions and throwing them into tibble
predict_test <- predict(besty_model, data_test, allow.new.levels = T) %>%
  nicify_predictions()

# joining with test set
df_test_eval <- data_test %>%
  join_preds(predict_test)

## Joining, by = "row_num"

# calculate RMSE
rmse_test <- Metrics::rmse(df_test_eval$CHI_MLU, df_test_eval$prediction)

rbind(rmse_model_train, rmse_test)

##                        [,1]
## rmse_model_train 0.2523631
## rmse_test        0.1703127
```

Exercise 2) Model Selection via Cross-validation (N.B: ChildMLU!)

One way to reduce bad surprises when testing a model on new data is to train the model via cross-validation.

In this exercise you have to use cross-validation to calculate the predictive error of your models and use this predictive error to select the best possible model.

- Use cross-validation to compare your model from last week with the basic model (Child MLU as a function of Time and Diagnosis, and don't forget the random effects!)
- (Tips): google the function "createFolds"; loop through each fold, train both models on the other folds and test them on the fold)
- Now try to find the best possible predictive model of ChildMLU, that is, the one that produces the best cross-validated results.
- Bonus Question 1: What is the effect of changing the number of folds? Can you plot RMSE as a function of number of folds?
- Bonus Question 2: compare the cross-validated predictive error against the actual predictive error on the test data

```
#- Create the basic model of ChildMLU as a function of Time and Diagnosis (don't
forget the random effects!). Also adding extra models with interactions.
m1 <- lmer(CHI_MLU ~ Visit + Diagnosis + (1|Child.ID),
                    data=df_train)
```

```
m2 <- lmer(CHI_MLU ~ Visit*Diagnosis + (1|Child.ID),
                     data=df_train)

m3 <- lmer(CHI_MLU ~ (Visit + Diagnosis + MOT_MLU + verbalIQ1)^2 + (1|Child.ID),
                     data=df_train)



#- Make a cross-validated version of the model. (Tips: google the function
"createFolds";  loop through each fold, train a model on the other folds and test
it on the fold)



folds <- createFolds(df_train$Child.ID, k=10)

result_df <- tibble(loop_num=numeric(), cv_num=numeric(), rmse=numeric())

for (j in 1:5) {
  i <- 1
  for (fold in folds) {
    traindat <- filter(df_train, !Child.ID %in% fold)
    testdat <- filter(df_train, Child.ID %in% fold)

    model <- update(besty_model, data=traindat)

    preds <- predict(besty_model, testdat, allow.new.levels=T) %>%
      nicify_predictions()

    eval_df <- testdat %>%
      join_preds(preds)
    result_df <- result_df %>%
      bind_rows(tibble(rmse=Metrics::rmse(eval_df$CHI_MLU,
                                          eval_df$prediction ),
                       cv_num = i,
                       loop_num = j))

    i <- i + 1
  }
}

## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"

result_df %>% summarize(mean_RMSE = mean(rmse))
```

```
## # A tibble: 1 x 1
##   mean_RMSE
##       <dbl>
## 1     0.244

#- Report the results and comment on them. (IN PDF)

#- Now try to find the best possible predictive model of ChildMLU, that is, the
one that produces the best cross-validated results.
model_list <- list("m1" = m1, "m2" = m2, "m3" = m3, "besty" = besty_model)


model_result_df <- tibble(model_name=character(), cv_num=numeric(),
rmse=numeric(), loop_num=numeric())

for (name in names(model_list)) {
  for (j in 1:5) {
    i <- 1
    for (fold in folds) {
      traindat <- filter(df_train, !Child.ID %in% fold)
      testdat <- filter(df_train, Child.ID %in% fold)

      model <- update(model_list[[name]], data=traindat)

      preds <- predict(model, testdat, allow.new.levels=T) %>%
        nicify_predictions()

      eval_df <- testdat %>%
        join_preds(preds)
      model_result_df <- model_result_df %>%
        bind_rows(tibble(rmse=Metrics::rmse(eval_df$CHI_MLU,
                                            eval_df$prediction),
                         model_name = name,
                         cv_num = i,
                         loop_num = j))

      i <- i + 1
    }
  }
}

## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
```

```
model_result_df %>%
  group_by(model_name) %>%
  summarise(rmse_cv = mean(rmse)) %>%
  arrange(rmse_cv)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 4 x 2
##   model_name rmse_cv
##   <chr>        <dbl>
## 1 m3           0.423
## 2 besty        0.431
## 3 m2           0.523
## 4 m1           0.626
```

Bonus Question 1: What is the effect of changing the number of folds? Can you plot RMSE as a function of number of folds?

```
num_folds <- c(2, 3, 4, 5, 6, 7, 8, 9, 10)

kfold_results <- tibble(rmse=numeric(),
                       num_folds = numeric(),
                       cv_num = numeric())

for (k in num_folds) {
  folds <- createFolds(df_train$Child.ID, k=k)
  i <- 1
  for (fold in folds) {
    traindat <- filter(df_train, !Child.ID %in% fold)
    testdat <- filter(df_train, Child.ID %in% fold)

    model <- update(m3, data=traindat)

    preds <- predict(model, testdat, allow.new.levels=T) %>%
      nicify_predictions()

    eval_df <- testdat %>%
      join_preds(preds)
    kfold_results <- kfold_results %>%
      bind_rows(tibble(rmse=Metrics::rmse(eval_df$CHI_MLU,
                                          eval_df$prediction),
                       num_folds = k,
                       cv_num = i))

    i <- i + 1
  }
}
```

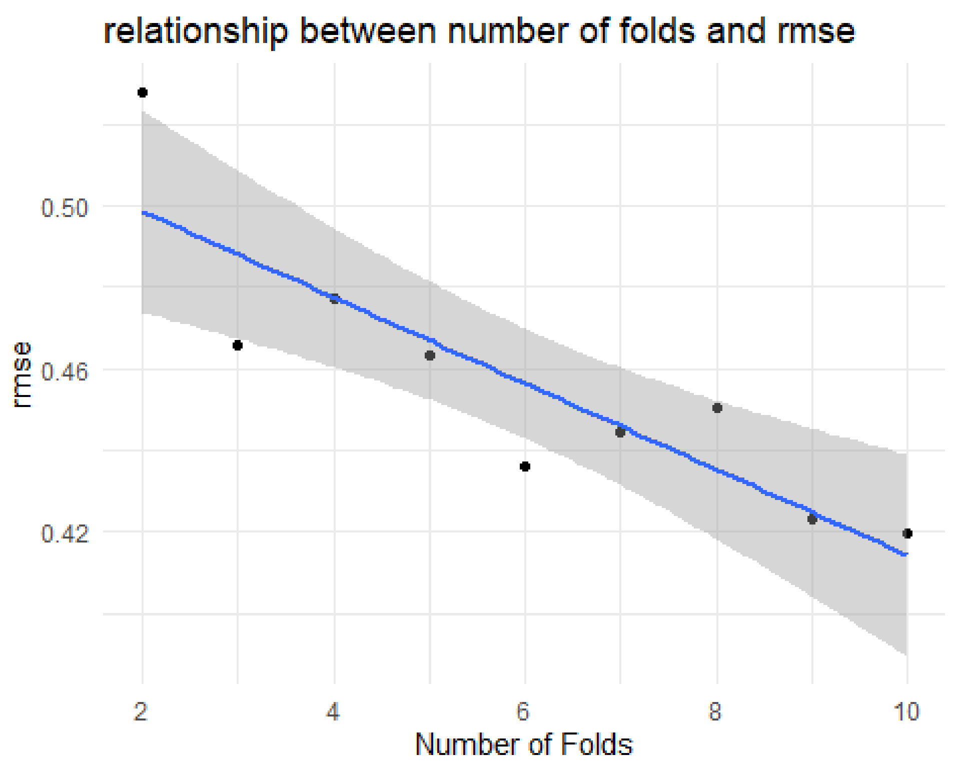
```
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
## Joining, by = "row_num"
```

```r
# Creating dataframe for plotting
viz_df <- kfold_results %>%
  group_by(num_folds) %>%
  summarise(num_folds = first(num_folds),
            rmse = mean(rmse))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
ggplot(viz_df, aes(x=num_folds, y=rmse)) +
  geom_point() +
  geom_smooth(method = "lm") +
  theme_minimal() +
  labs(title = "relationship between number of folds and rmse",
       x = "Number of Folds")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Bonus Question 2: compare the cross-validated predictive error against the actual predictive error on the test data

```
test_set_results <- tibble(model_name = names(model_list),
                           test_rmse = 0)
# Creating test Loop
for (model_name in names(model_list)) {
  model <- model_list[[model_name]]
  test_preds <- predict(model, data_test, allow.new.levels = T) %>%
    nicify_predictions()
  eval_test <- data_test %>%
    join_preds(test_preds)

  test_rmse <- Metrics::rmse(eval_test$CHI_MLU, eval_test$prediction)

  test_set_results <- test_set_results %>%
    rows_update(tibble(model_name = model_name,
                       test_rmse = test_rmse))


}

## Joining, by = "row_num"

## Matching, by = "model_name"
## Joining, by = "row_num"

## Matching, by = "model_name"

## Joining, by = "row_num"

## Matching, by = "model_name"

## Joining, by = "row_num"

## Matching, by = "model_name"

# Joinign and plotting
rmse_viz_df <- model_result_df %>%
  group_by(model_name) %>%
  summarise(cv_rmse = mean(rmse)) %>%
  left_join(test_set_results) %>%
  gather(key = "result_type", value = "rmse", -model_name)

## `summarise()` ungrouping output (override with `.groups` argument)

## Joining, by = "model_name"

compare_plot <- ggplot(rmse_viz_df, aes(model_name, rmse)) +
  geom_bar(aes(fill = result_type), position="dodge",
           stat="identity") +
  theme_minimal() +
  labs(title = "rmse: CV vs Test set", x = "Model Name")

ggsave("rmse_compare.png", compare_plot)

## Saving 5 x 4 in image
```

Exercise 3) Assessing the single child

Let's get to business. This new kiddo - Bernie - has entered your clinic. This child has to be assessed according to his group's average and his expected development.

Bernie is one of the six kids in the test dataset, so make sure to extract that child alone for the following analysis.

You want to evaluate:

- how does the child fare in ChildMLU compared to the average TD child at each Visit? Define the distance in terms of absolute difference between this Child and the average TD.

```
non_normalized_data <- CleanUpData(demo_test, lu_test, token_test)

bernie_mlu <- non_normalized_data %>%
  filter(Child.ID == "Bernie") %>%
  pull(CHI_MLU)
```

```
mean_non_normalized_MLU <- read_csv("data_clean.csv") %>%
  filter(Diagnosis == "TD") %>%
  group_by(Visit) %>%
  summarise(mean_MLU = mean(CHI_MLU, na.rm=T)) %>%
  mutate(bernie_score = bernie_mlu - mean_MLU)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Ethnicity = col_character(),
##   Diagnosis = col_character(),
##   Gender = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
mean_non_normalized_MLU
```

```
## # A tibble: 6 x 3
##   Visit mean_MLU bernie_score
##   <dbl>    <dbl>        <dbl>
## 1     1     1.31        0.674
## 2     2     1.76        0.782
## 3     3     2.23        1.12
## 4     4     2.73        0.448
## 5     5     2.97        0.203
## 6     6     2.93        0.521
```

- how does the child fare compared to the model predictions at Visit 6? Is the child below or above expectations? (tip: use the predict() function on Bernie's data only and compare the prediction with the actual performance of the child)

```
bernie_data <- data_test %>%
  filter(Child.ID == "Bernie") %>%
  mutate(row_num = row_number())



bernie_preds <- bernie_data %>%
  # The best model from cv excluding the disaster
  predict(m3, ., allow.new.levels=T) %>%
  nicify_predictions()
```

```
bernie_data %>%
  left_join(bernie_preds) %>%
  filter(Visit == max(Visit)) %>%
  select(CHI_MLU, prediction) %>%
  mutate(abs_dif = prediction - CHI_MLU)

## Joining, by = "row_num"

##    CHI_MLU prediction    abs_dif
## 1 1.500245   1.360947 -0.1392982
```

[HERE GOES YOUR ANSWER]

OPTIONAL: Exercise 4) Model Selection via Information Criteria

Another way to reduce the bad surprises when testing a model on new data is to pay close attention to the relative information criteria between the models you are comparing. Let's learn how to do that!

Re-create a selection of possible models explaining ChildMLU (the ones you tested for exercise 2, but now trained on the full dataset and not cross-validated).

Then try to find the best possible predictive model of ChildMLU, that is, the one that produces the lowest information criterion.

```
pacman::p_load(broom.mixed)

m1 <- lmerTest::lmer(CHI_MLU ~ Visit + Diagnosis + (1|Child.ID),
                    data=df_train, REML = F)

m2 <- lmerTest::lmer(CHI_MLU ~ Visit*Diagnosis + (1|Child.ID),
                    data=df_train, REML = F)

m3 <- lmerTest::lmer(CHI_MLU ~ (Visit + Diagnosis + MOT_MLU + verbalIQ1)^2 +
(1|Child.ID),
                    data=df_train, REML = F)

besty_model <- lmerTest::lmer(CHI_MLU ~ Visit + Ethnicity + Diagnosis + Gender +
Age + Socialization +
    MOT_MLU + types_MOT + types_CHI +
    tokens_MOT + tokens_CHI + (1 | Child.ID) + Visit:Diagnosis +
    Visit:Gender + Visit:Socialization + Ethnicity:Age + Ethnicity:types_MOT +
    Ethnicity:types_CHI + Ethnicity:tokens_MOT + Diagnosis:Socialization +
Gender:Age +
    Gender:Socialization + Gender:types_CHI + Gender:tokens_CHI +
    Age:Socialization + Age:MOT_MLU  +
    Socialization:tokens_MOT + MOT_MLU:tokens_MOT + MOT_MLU:tokens_CHI +
    types_MOT:types_CHI + types_CHI:tokens_MOT + tokens_MOT:tokens_CHI,
data=df_train, REML = F)


model_list <- list("m1" = m1, "m2" = m2, "m3" = m3, "besty" = besty_model)

# glancing at the stuff
get_relevant_metrics <- function(model_name) {
  model_list[[model_name]] %>%
    glance %>%
    mutate(name = model_name) %>%
    select(name, AIC, BIC)
}


summarised_cv <- model_result_df %>%
  group_by(model_name) %>%
  summarise(cv_rmse = mean(rmse))

## `summarise()` ungrouping output (override with `.groups` argument)

IC_df <- map_df(names(model_list), get_relevant_metrics) %>%
  arrange(AIC) %>%
  left_join(summarised_cv, by = c("name" = "model_name"))

IC_df

## # A tibble: 4 x 4
##   name    AIC   BIC cv_rmse
##   <chr> <dbl> <dbl>   <dbl>
## 1 besty  353.  484.   0.431
## 2 m3     528.  578.   0.423
## 3 m2     642.  665.   0.523
## 4 m1     711.  731.   0.626
```

Note: Even though the performance of besty is sub-par on the hold-out data, it still performs vastly better on information criterion measures - shows the issues of those

- Bonus question for the optional exercise: are information criteria correlated with cross-validated RMSE? That is, if you take AIC for Model 1, Model 2 and Model 3, do they co-vary with their cross-validated RMSE?

OPTIONAL: Exercise 5): Using Lasso for model selection

Welcome to the last secret exercise. If you have already solved the previous exercises, and still there's not enough for you, you can expand your expertise by learning about penalizations. Check out this tutorial: http://machinelearningmastery.com/penalized-regression-in-r/ and make sure to google what penalization is, with a focus on L1 and L2-norms. Then try them on your data!

We chose not to do this, as the penalized regression mainly works for ordinary linear models (not mixed effects models). Doing otherwise would take tooo much tweaking to be worth it :))