Multiple-Choice Online Causal Comprehension Assessment (MOCCA): Reimagining Question Readability

Anwesha Guha

12/9/2021

Research Problem

This project serves as exploratory analysis and introduction to model-building using data from the Multiple-Choice Online Causal Comprehension Assessment (MOCCA). MOCCA stems from the original 40-item test to test reading comprehension and has been converted to three forms for each grade in grades 3-5. This results in 360 unique test questions in total, which we use in this project.

There is much research supporting and validating the development of this assessment, according to its website. MOCCA is a meaningful diagnostic assessment of reading comprehension that helps teachers understand not only whether students struggle with comprehension, but also the underlying reasons for their struggles (Biancarosa et al., 2019; Liu et al., 2019). It has been fully validated as a diagnostic assessment of reading comprehension with excellent reliability (Biancarosa et al., 2019; Davison et al., 2018; Su & Davison, 2019). There is also preliminary evidence of its validity as a screener for reading comprehension difficulties (Biancarosa et al., 2019). Because of this, understanding more deeply the readability of the test items beyond the Flesch-Kincaid score used in the model is meaningful as MOCCA continues to expand in the future. Findings can also be used to tune similar testing items in the field.

Reading score is the primary outcome of interest. This analysis will manipulate the MOCCA text data using the Roberta-base model in order to predict the Flesch-Kincaid reading score. This reading score will be predicted using the item response theory (IRT) parameters and 768 dimensions that is extracted from each story item.

The work here will serve as an exploration and foundation. Much of IRT predictive analysis rely on the simplicity of the Flesch-Kincaid model. While this may so far be sufficient when considering readability as a small characteristic of the item measure in IRT, there is potential for it to be used in more careful and complex ways in the future, including by providing a more complex measure of text readability for its items to better serve students at their appropriate academic levels.

Description of Data

Flesch-Kincaid Score

Developed by Rudolf Flesch and J. Peter Kincaid, the Flesch-Kincaid readability scores are the most widely used measures of readability. Reading ease is evaluated using the following formula:

$$Score = 206.835 - 1.015(words/sentences) - 84.6(syllables/words)$$

On average, most text and media that people consume average at about a 7th-grade level. If the text is more complicated, it may alienate some members of the audience. If the text scores at a lower level, people

may assume the content is not valuable. These have valuable implications in a data, text, and media-driven world.

However, in this testing context, the Flesch-Kincaid score is used a bit differently. Since the objective of the text in this data is to be used to test reading comprehension and the associated common errors and misdirections, the Flesch-Kincaid score makes sure the test items are appropriate for students. As a result, the FK scores will generally correlate with the grade level and term in which students are testing. This is the `form` column in the dataset.

MOCCA

The data itself was compiled from MOCCA test item metrics. MOCCA test items were originally broken down by sentence, with the sixth sentence missing, out of seven sentences. Each is coded as a separate column in the data.

The missing sentence, which is represented as one of the three answer choices, is termed the `cci` (causally coherent inference). The two distractors are labeled `par` (paraphrase) and `lcn` (lateral connection, now referred to as elaboration).

The entire story was reconstructed using the six sentences and `cci` to produce the `story` except. The Flesch-Kincaid readability score is constructed from this `story` item.

IRT Parameters

Alongside the text from each of these stories, certain IRT parameters were provided. These parameters on which items are characterized include their *difficulty* (known as "location" for their location on the difficulty range); *discrimination* (slope or correlation), representing how steeply the rate of success of individuals varies with their ability; and a *pseudoguessing parameter*, characterizing the (lower) asymptote at which even the least able persons will score due to guessing (for instance, 25% for pure chance on a multiple choice item with four possible responses). They are included for the reading comprehension and propensity dimensions.

These specific variables included below.

`dim1_p` = Proportion correct in the calibration sample for the reading comprehension dimension

`dim1_r` = Item correlation for the reading comprehension dimension

`dim1_a` = Item discrimination for the reading comprehension dimension. Steeper slopes at particular level of the traits indicate that it is more discriminative than levels of the traits with gentler slopes.

`dim1_b` = Item difficulty for the reading comprehension dimension. The difficulty parameter is used to describe how difficulty an item is to achieve a 0.5 probability of correct response at a given ability. Therefore, if the 50% chance of correctly answering an item is at the right side of the plot, then it means that a higher level of ability is needed to achieve that goal.

`dim1_c` = Guessing parameter for the reading comprehension dimension. Guessing parameter is likely compared to threshold of 1/3 as the pseudo-chance-level parameter

`dim2_a` = Item discrimination estimation of the process propensity dimension

`dim2_b` = Item difficulty estimation of the process propensity dimension

`dim2_c` = Guessing parameter estimation of the process propensity dimension

Constructed parameters

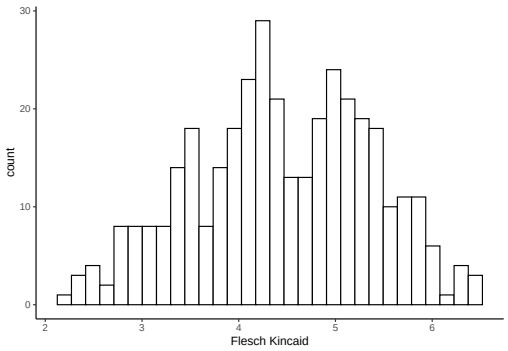
The RoBERTa model was pre-trained using BookCorpus, English Wikipedia, CC-News, OpenWebText, and Stories dataset. Notably, because this model contains unfiltered content, it can have biased predictions.

Using the RoBERTa base model, 768-hidden dimensions were extracted for each story/test item.

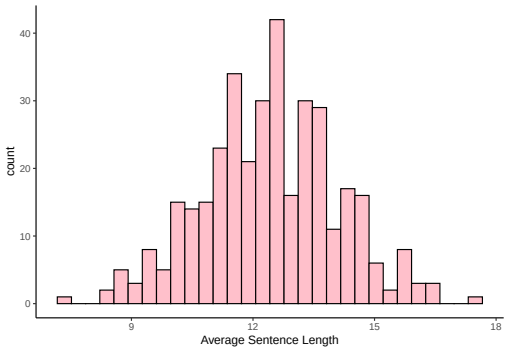
These dimensions can be used to fine-tune the prediction model but are masked and not able to be interpreted in themselves.

Data Characteristics

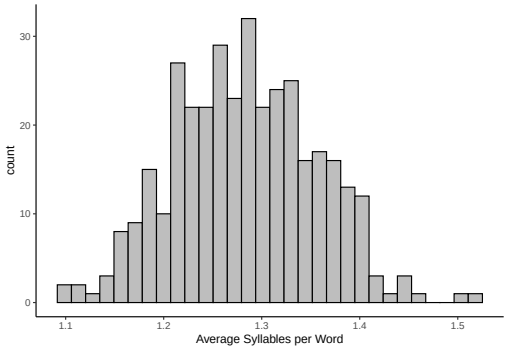
The following are some descriptive characteristics of this dataset. There are no missing items in this dataset.



Average Flesch-Kincaid score is 4.4468687.

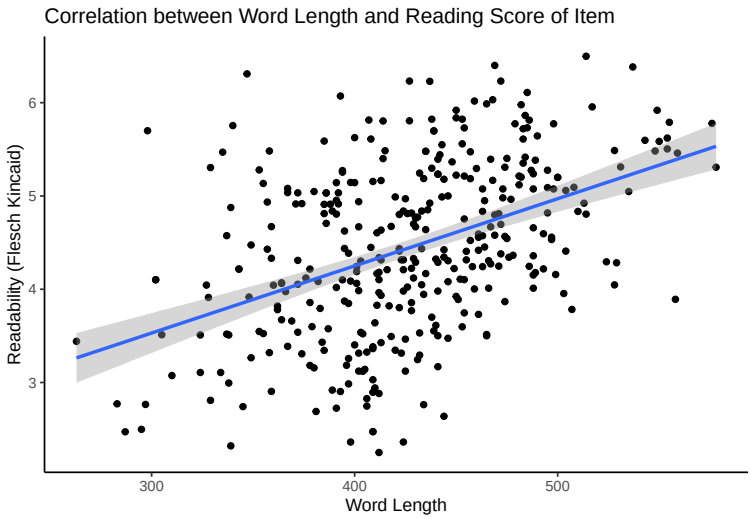


The average sentence length is 12.4343254.



The average syllables per word is 1.2832293.

A simple linear regression predicting the Flesch Kincaid reading score by word length is below.



Using a simple linear regression, we find that word length alone, while a significant predictor, only explains 19.18% of the variance in Flesch-Kincaid reading score, $\hat{\beta}_1$=0.0072, $SE(\hat{\beta}_1)$=0.0008, $t(358)$=9.219, p<.001. The highed the average word length, the more difficult it is to read.

Using machine learning, we will try three different modeling approaches below.

Description of Models

The continuous outcome of Flesch-Kincaid score will be predicted using three models: 1) linear regression without any regularization, 2) linear regression using the ridge penalty, and 3) linear regression using the lasso penalty.

For each model, the root mean squared error (RMSE) will be calculated to evaluate model performance.

Model Preparation

First, we use the `recipes` package to create a recipe for the dataset. This recipe will do the following:

- assigns the first column (`flesch_kincaid`) as outcome and everything else as predictors,
- removes any variable with zero variance or near-zero variance,
- impute the missing values using the mean (if any),
- standardize all variables,
- and removes variables highly correlated with one another (>.9).

```
blueprint <- recipe(x     = mocca,
                    vars  = colnames(mocca),
                    roles = c("outcome", rep('predictor',780))) %>%
  step_zv(all_numeric()) %>%
  step_nzv(all_numeric()) %>%
  step_impute_mean(all_numeric()) %>%
  step_normalize(all_numeric_predictors()) %>%
  step_corr(all_numeric(),threshold=0.9)
```

In order to obtain a realistic measure of model performance, we will split the data into two subsamples: training and test datasets. We will use a 80-20 split.

```
set.seed(12022021)  # for reproducibility

loc      <- sample(1:nrow(mocca), round(nrow(mocca) * 0.8))
mocca_tr  <- mocca[loc, ]
mocca_te  <- mocca[-loc, ]
```

We will first train the blueprint using the training dataset, and then bake it for both training and test datasets.

```
prepare <- prep(blueprint,
                training = mocca_tr)
```

```
baked_tr <- bake(prepare, new_data = mocca_tr)

baked_te <- bake(prepare, new_data = mocca_te)
```

We will use these values and preparation in all three models below.

Model 1

Linear regression without regularization is the starting model. Unlike the simple linear regression above, this model will serve to optimize predictors to create the most accurate model with the best number of predictors and the least amount of error using 10-fold cross validation. However, due to having 780 predictors and a relatively small sample, overfitting is likely.

First, the model will need to be prepared and trained.

```
# Randomly shuffle the data
mocca_tr = mocca_tr[sample(nrow(mocca_tr)),]

# Create 10 folds with equal size
folds = cut(seq(1,nrow(mocca_tr)),breaks=10,labels=FALSE)

# Create the list for each fold
my.indices <- vector('list',10)
      for(i in 1:10){
        my.indices[[i]] <- which(folds!=i)
      }

cv <- trainControl(method = "cv",
                   index  = my.indices)

# Train Model 1
caret_mod1 <- caret::train(blueprint,
                           data      = mocca_tr,
                           method    = "lm",
                           trControl = cv)

caret_mod1
```

```
## Linear Regression
##
## 288 samples
## 780 predictors
##
## Recipe steps: zv, nzv, impute_mean, normalize, corr
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 259, 259, 259, 260, 259, 259, ...
## Resampling results:
##
##   RMSE      Rsquared    MAE
##   9.485366  0.07364104  7.364072
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Then, apply the model to the MOCCA test data.

```
predicted_te1 <- predict(caret_mod1, mocca_te)
```

Finally, calculate the evaluation metrics for this model.

```
rsq_te1 <- cor(mocca_te$flesch_kincaid,predicted_te1)^2
rsq_te1
```

```
## [1] 2.769598e-05
```

```
mae_te1 <- mean(abs(mocca_te$flesch_kincaid - predicted_te1))
mae_te1
```

```
## [1] 5.780729
```

```
rmse_te1 <- sqrt(mean((mocca_te$flesch_kincaid - predicted_te1)^2))
rmse_te1
```

```
## [1] 7.613469
```

Model 2

Model 2 and Model 3 will introduce regularization to constrain the size of regression coefficients to reduce their sampling variation and, thus, reduce the variance of each model prediction.

We will first start by tuning the λ hyperparameter (α=0). While many ranges were tried, we ultimately used the following grid:

```
grid <- data.frame(alpha = 0, lambda = seq(0,3,.1))
grid
```

```
##    alpha lambda
## 1      0    0.0
## 2      0    0.1
```

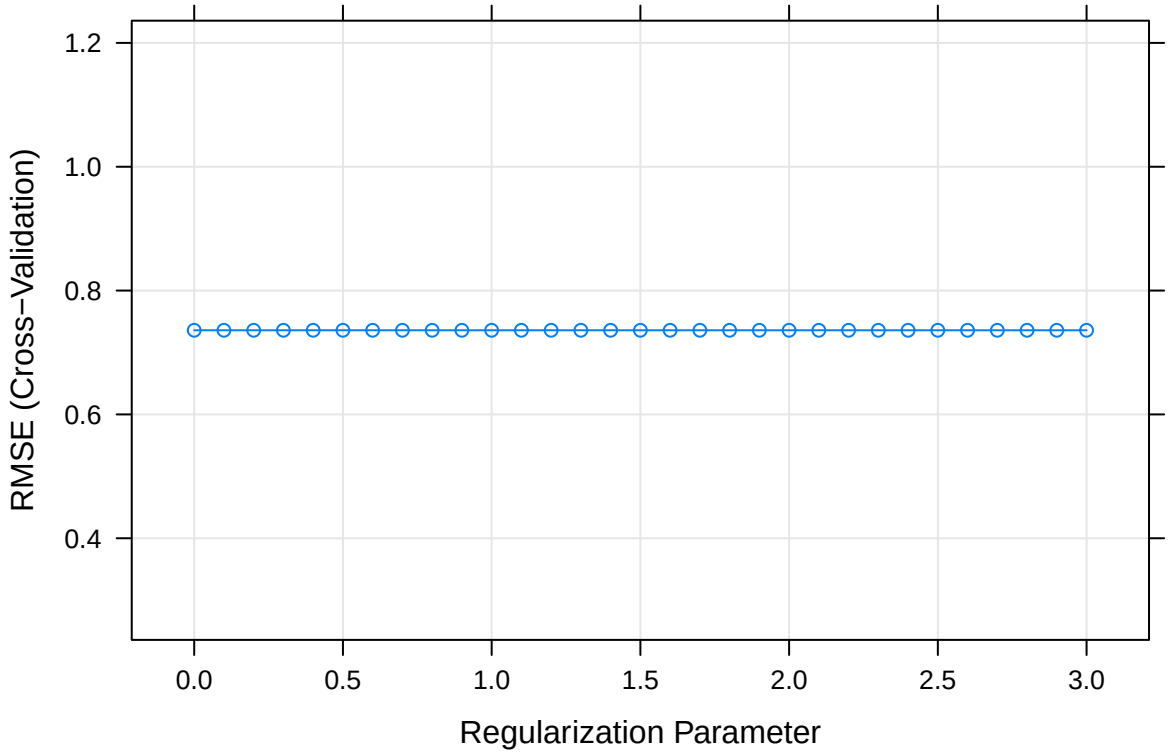
```
## 3      0    0.2
## 4      0    0.3
## 5      0    0.4
## 6      0    0.5
## 7      0    0.6
## 8      0    0.7
## 9      0    0.8
## 10     0    0.9
## 11     0    1.0
## 12     0    1.1
## 13     0    1.2
## 14     0    1.3
## 15     0    1.4
## 16     0    1.5
## 17     0    1.6
## 18     0    1.7
## 19     0    1.8
## 20     0    1.9
## 21     0    2.0
## 22     0    2.1
## 23     0    2.2
## 24     0    2.3
## 25     0    2.4
## 26     0    2.5
## 27     0    2.6
## 28     0    2.7
## 29     0    2.8
## 30     0    2.9
## 31     0    3.0
```

```
ridge <- caret::train(blueprint,
                      data      = mocca_tr,
                      method    = "glmnet",
                      trControl = cv,
                      tuneGrid  = grid)
```

```
ridge$bestTune
```

```
##    alpha lambda
## 31     0      3
```

```
plot(ridge)
```



In this particular case, there is not observable change between the values. So, we go ahead and apply model 2 to the MOCCA test data.

```
predict_te_ridge <- predict(ridge, mocca_te)
```

Next, calculate the evaluation metrics for this model.

```
rsq_te_ridge <- cor(mocca_te$flesch_kincaid, predict_te_ridge)^2
rsq_te_ridge
```

```
## [1] 0.4488322
```

```
mae_te_ridge <- mean(abs(mocca_te$flesch_kincaid - predict_te_ridge))
mae_te_ridge
```

```
## [1] 0.5457634
```

```
rmse_te_ridge <- sqrt(mean((mocca_te$flesch_kincaid - predict_te_ridge)^2))
rmse_te_ridge
```

```
## [1] 0.6728178
```

Model 3

Finally, for model 3, we will tune the λ hyperparameter (α=1).

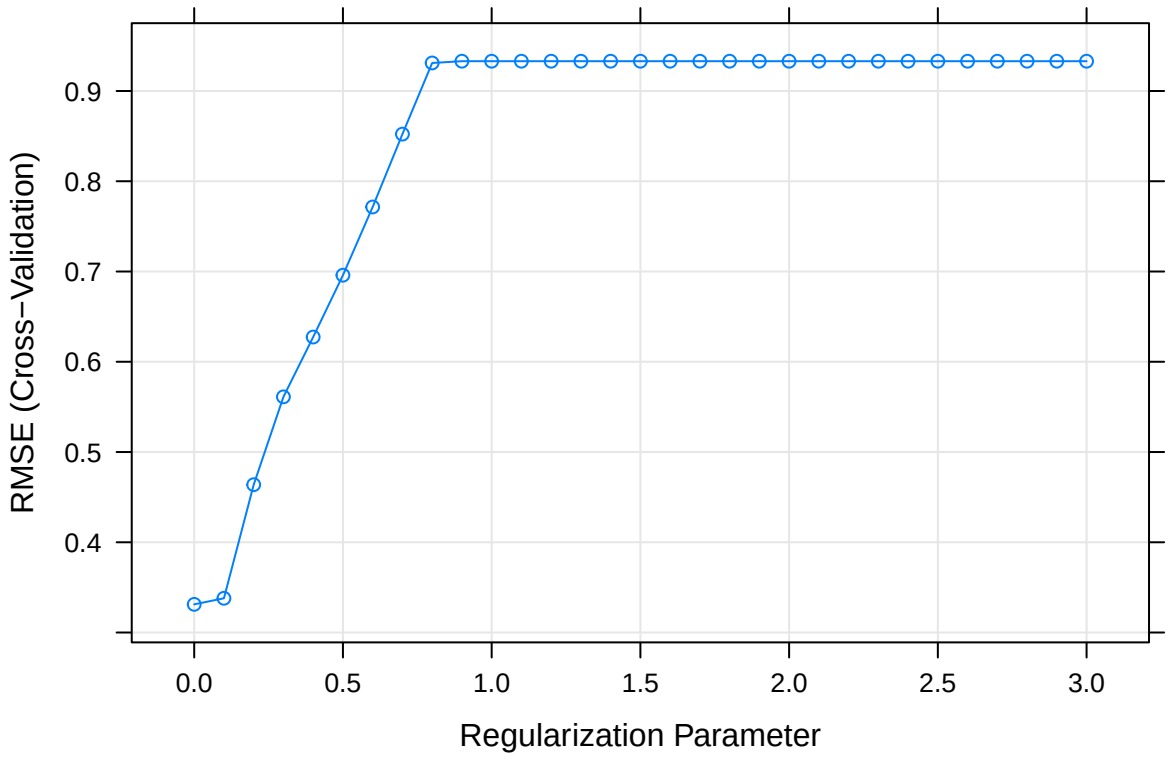
```
grid2 <- data.frame(alpha = 1, lambda = seq(0,3,.1))

lasso <- caret::train(blueprint,
                      data      = mocca_tr,
                      method    = "glmnet",
                      trControl = cv,
                      tuneGrid  = grid2)
```

```
## Warning in train_rec(rec = x, dat = data, info = trainInfo, method = models, :
## There were missing values in resampled performance measures.
```

```
lasso$bestTune
```

```
##   alpha lambda
## 1     1      0
```

```
plot(lasso)
```



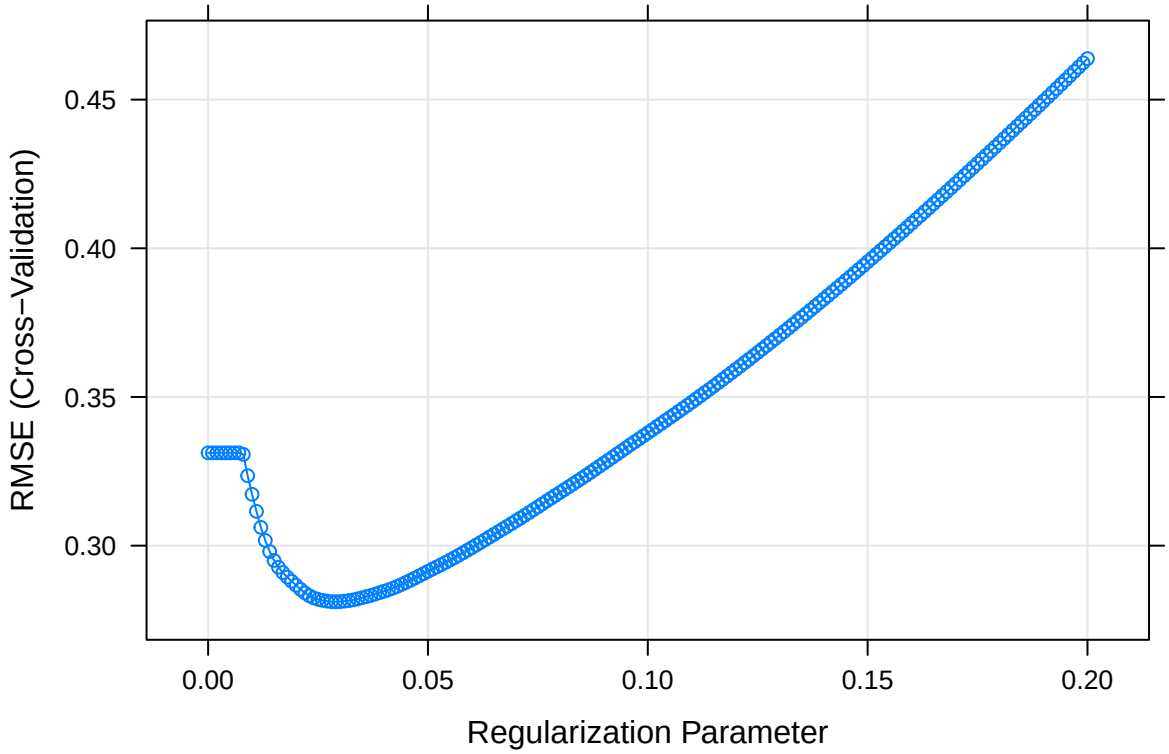
It appears that the lambda value is optimal between 0.0 and 0.2. We will run the model again with a different grid to get a more accurate value.

```
grid3 <- data.frame(alpha = 1, lambda = seq(0,.2,.001))

lasso2 <- caret::train(blueprint,
                      data      = mocca_tr,
                      method    = "glmnet",
                      trControl = cv,
```

```
                        tuneGrid  = grid3)

lasso2$bestTune
```

```
##    alpha lambda
## 30     1  0.029
```

```
plot(lasso2)
```



Now, apply model 3 to the MOCCA test data.

```
predict_te_lasso<- predict(lasso2, mocca_te)
```

Calculate the evaluation metrics for Model 3.

```
rsq_te_lasso <- cor(mocca_te$flesch_kincaid, predict_te_lasso)^2
rsq_te_lasso
```

```
## [1] 0.8498041
```

```
mae_te_lasso <- mean(abs(mocca_te$flesch_kincaid - predict_te_lasso))
mae_te_lasso
```

```
## [1] 0.2735025
```

Table 1: Model Fit

	RMSE	MAE	R-sq
Linear Regression	0.000	5.781	0.000
Lin. Reg. with Ridge	0.673	0.546	0.449
Lin. Reg. with Lasso	0.331	0.274	0.850

```
rmse_te_lasso <- sqrt(mean((mocca_te$flesch_kincaid - predict_te_lasso)^2))
rmse_te_lasso
```

```
## [1] 0.3308614
```

Model Fit

Discussion/Conclusion

Next steps involve using the readability score calculated using the Kaggle readability dataset to and apply to MOCCA for a more complex understanding of MOCCA test item readability. MOCCA expands into computer-adaptive testing.

Discuss and summarize what you learned. Which variables were the most important in predicting your outcome? Was this expected or surprising? Were different models close in performance, or were there significant gaps in performance from different modeling approaches? Are there practical/applied findings that could help the field of your interest based on your work? If yes, what are they?

Future models may continue to tune using decision trees, bagged trees and random forests.

Word count: 2182