Skills Network

Refine the Baseline Regression Models

Estimated time needed: **120** minutes

Lab Overview:

Now you have built a baseline regression model with some relatively good RMSE and R-squared reported values. However, we could still improve it by using methods like adding polynomial and interaction terms, regularization, and so on.

In this lab, you will be asked to continue using `tidymodels` to improve the performance of baseline model:

- **TASK: Add polynomial terms**
- **TASK: Add interactions terms**
- **TASK: Add regularizations terms**
- **TASK: Experiment to search for improved models**

Let's start!

The tidyverse and the tidymodels packages can be used to produce high quality statistical and machine learning models. The Tidyverse library is a useful tool that provides various tools for data visualization, data manipulation, and read various datasets into a data frame; our Jupyter notebook platforms have a built-in Tidymodels, Tidyverse and rlang libraries, so we do not need to install these packages prior to loading library. However, if you decide to run this lab on your RStudio Desktop locally on your machine, you can remove the commented lines of code to install these packages before loading.

In [ ]:

```
# Uncomment to install packages if running locally
# install.packages("tidyverse")
# install.packages("tidymodels")
```

In [ ]:

```
library("tidymodels")
library("tidyverse")
library("stringr")
```

The processed Seoul bike sharing dataset `seoul_bike_sharing_converted_normalized.csv` , includes the converted indicator variables, and the numerical variables have been normalized. Let's read it as a dataframe first:

In [ ]:

```
# Dataset URL
dataset_url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo
bike_sharing_df <- read_csv(dataset_url)
spec(bike_sharing_df)
```

We won't be using the `DATE` column, because 'as is', it basically acts like an data entry index. (However, given more time, we could use the `DATE` colum to create a 'day of week' or 'isWeekend' column, which we might expect has an affect on preferred bike rental times.) We also do not need the `FUNCTIONAL DAY` column because it only has one distinct value remaining (`YES`) after missing value processing.

In [ ]:

```
bike_sharing_df <- bike_sharing_df %>%
                   select(-DATE, -FUNCTIONING_DAY)
```

Define a linear regression model specification.

In [ ]:

```
lm_spec <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")
```

Split the data into training and testing datasets.

In [ ]:

```
set.seed(1234)
data_split <- initial_split(bike_sharing_df, prop = 4/5)
train_data <- training(data_split)
test_data <- testing(data_split)
```

Now we are ready to refine the previous baseline regression model.

TASK: Add polynomial terms

Linear regression models are the most suitable models to capture the linear correlations among variables. However, in real world data, many relationships may be non-linear.

For example, the correlation between `RENTED_BIKE_COUNT` and `TEMPERATURE` does not look like linear:

In [ ]:

```
ggplot(data = train_data, aes(RENTED_BIKE_COUNT, TEMPERATURE)) +
    geom_point()
```

One solution to handle such nonlinearity is using polynomial regression by adding polynomial terms. As shown before, higher order polynomials are better than the first order polynomial.

In [ ]:

```
# Plot the higher order polynomial fits
ggplot(data=train_data, aes(RENTED_BIKE_COUNT, TEMPERATURE)) +
    geom_point() +
    geom_smooth(method = "lm", formula = y ~ x, color="red") +
    geom_smooth(method = "lm", formula = y ~ poly(x, 2), color="yellow") +
    geom_smooth(method = "lm", formula = y ~ poly(x, 4), color="green") +
    geom_smooth(method = "lm", formula = y ~ poly(x, 6), color="blue")
```

OK, let's add some higher order polynomials of important variables to the regression models

TODO: Fit a linear regression model `lm_poly` with higher order polynomial terms on the important variables (larger coefficients) found in the baseline model

In [ ]:

```
# Fit a linear model with higher order polynomial on some important variables

# #HINT: Use ploy function to build polynomial terms, lm_poly <- RENTED_BIKE_COU
```

In [ ]:

```
# Print model summary

# summary(lm_poly$fit)
```

TODO: Make predictions on test dataset using the `lm_poly` models

In [ ]:

```
# Use predict() function to generate test results for `lm_poly`
```

Another minor improvement we could do here is to convert all negative prediction results to zero, because we can not have negative rented bike counts

In [ ]:

```
# e.g., test_results[test_results<0] <- 0
```

Now, calculate R-squared and RMSE for the test results generated by `lm_ploy` model

In [ ]:

```
# Calculate R-squared and RMSE from the test results
```

If you include all variables, and additionally include some of the more important ones as higher order poly terms, then you should notice improved `R-squared` and `RMSE` values.

TASK: Add interaction terms

In real-world scenarios, in addition to non-linear relationships between response variables and predictor variables, you may also encounter relationships among variables called `interaction effects`.

For example, the effect of predictor variable `TEMPERATURE` on `RENTED_BIKE_COUNT` may also depend on other variables such as `HUMIDITY`, `RAINFALL`, or both (they `interact`) and the effect of `SEASON` on `RENTED_BIKE_COUNT` may also depend on `HOLIDAY`, `HOUR`, or both.

To capture such interaction effects, we could add some interaction terms such as `RAINFALL*HUMIDITY` to the regression model, similar to what we did with polynominal terms. In this task, you will explore and conduct some experiments to search for interaction terms which will improve the model performance.

TODO: Try adding some interaction terms to the previous polynomial models.

In [ ]:

```
# Add interaction terms to the poly regression built in previous step

# HINT: You could use `*` operator to create interaction terms such as HUMIDITY*
# RENTED_BIKE_COUNT ~ RAINFALL*HUMIDITY ...
```

In [ ]:

```
# Print model summary
```

In [ ]:

```
# Calculate R-squared and RMSE for the new model to see if performance has impro
```

TASK: Add regularization

In previous tasks, you were asked to add polynominal and interaction terms to the model, aiming to capture nonlinearity and interaction effects between the predictor variables. Hopefully, your updated models have better R-squared and RMSE values.

However, adding these terms makes your model more complicated, more difficult to explain, and more likely to suffer from overfitting. To overcome these issues, one solution is to add regularization terms to your models.

When building the baseline model, we used the basic `lm` engine. In this task, you will use a more advanced and generalized `glmnet` engine. It provides a generalized linear model with Lasso, Ridge, and Elastic Net regularizations.

In general, using `glmnet` can enhance your models in the following ways:

- Address overfitting issues by shrinking the coefficients
- Address predictor variable colinearity by selecting only one variable from each group of colinear variables (by shrinking their coefficients to zero)
- Make your models more interpretable due to simplification (fewer variables in the outcome models)

Now, let's switch our regression engine to `glmnet`

TODO: Define a linear regression model specification `glmnet_spec` using `glmnet` engine

In [ ]:

```
# HINT: Use linear_reg() function with two parameters: penalty and mixture
# - penalty controls the intensity of model regularization
# - mixture controls the tradeoff between L1 and L2 regularizations

# You could manually try different parameter combinations or use grid search to
```

Fit a glmnet model called `lm_glmnet` using the `fit()` function. For the formula part, keep the polynominal and interaction terms you used in the previous task.

In [ ]:

```
install.packages('glmnet')
```

In [ ]:

```
library('glmnet')
```

In [ ]:

```
# Fit a glmnet model using the fit() function
```

In [ ]:

```
# Report rsq and rmse of the `lm_glmnet` model
```

TASK: Experiment to search for improved models

Now you understand some of the methods that you can use to try to improve your models.

TODO: Experiment by building and testing at least five different models. For each of your experiments, include polynomial terms, interaction terms, and one of the three regularizations we introduced.

In [ ]:

```
# Build at least five different models using polynomial terms, interaction terms

# Save their rmse and rsq values
```

In [ ]:

```
# Report the best performed model in terms of rmse and rsq
```

Here are the performance requirements for your best model:

- The RMSE should be less than 330 (rougly 10% of the max value in test dataset)
- R-squared should be greater than 0.72

TODO: Visualize the saved RMSE and R-squared values using a grouped barchart

In [ ]:

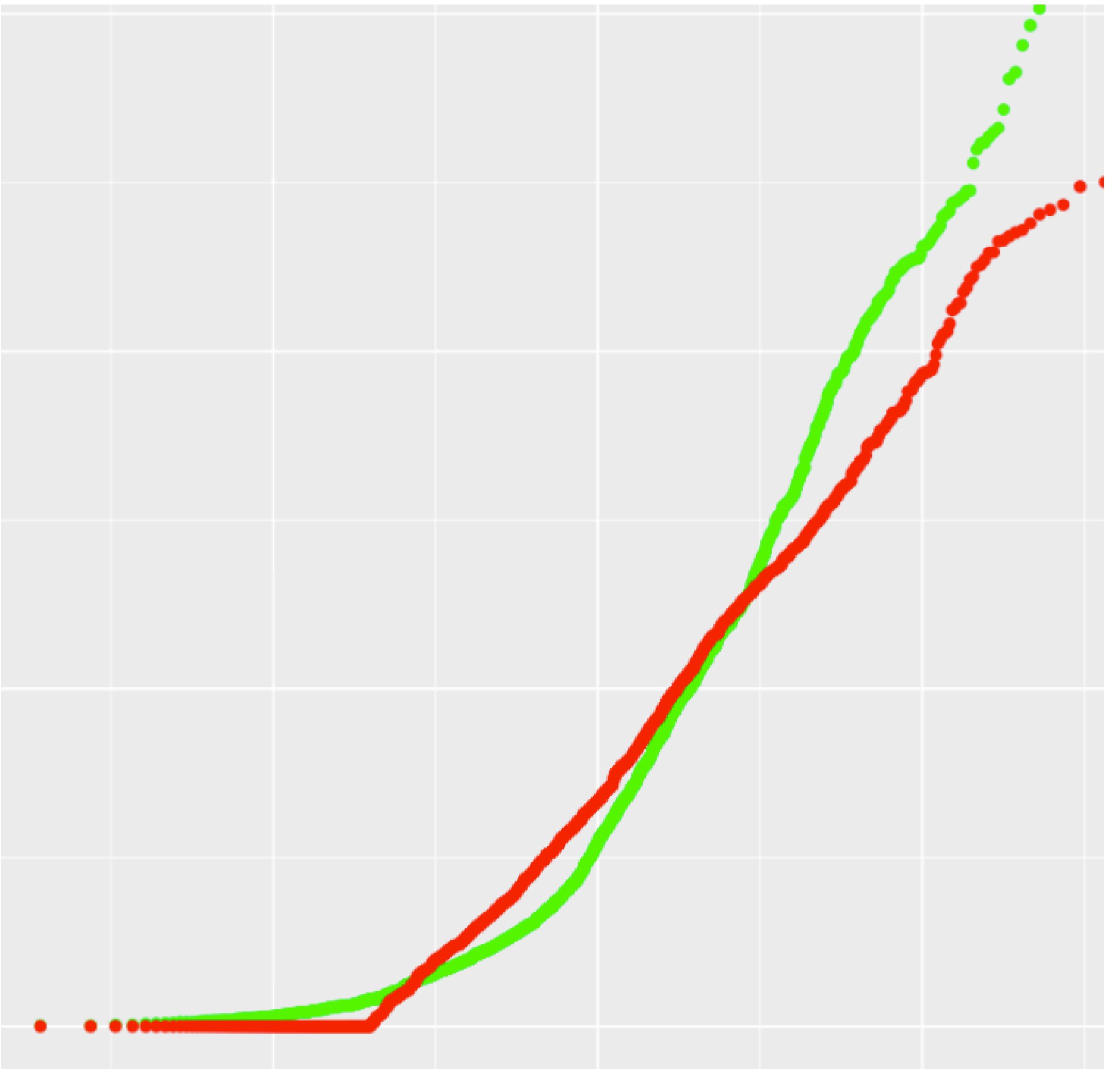
```
# HINT: Use ggplot() + geom_bar()
```

TODO: Create a Q-Q plot by plotting the distribution difference between the predictions generated by your best model and the true values on the test dataset.

In [ ]:

```
# HINT: Use ggplot() +
    # stat_qq(aes(sample=truth), color='green') +
    # stat_qq(aes(sample=prediction), color='red')
```

One example of such Q-Q plot may look like this:



More model improvment methods beyond this course

In addition to the methods mentioned in this lab and previous data analysis courses, you could also explore to try the following methods yourself to see if they could improve model performance:

- Remove potential redundant variables. If two variables have extremly high correlated, it is possible that they are redundant and could be removed from the model to improve the performance.
- Remove some outliers. Linear regression models are very sensitive to outliers, you could try to remove some outliers to see if it would improve performance
- Apply logarithm transformation. In case variable distributions are not normal distribution such as log-normal distribution, you could apply logaritm transformation on the variable to make them more look like normal distribution. In addition, logarithm transformation helps capture the non-linear relationships.

If you have time, you could research and try more methods by searching related research papers/articles, discussion forums, etc. If you know how to use other machine learning models with `Tidymodels` such as Neural Networks, Tree models, or Boosting models, you can also try and compare them with the linear regression models.

Next Steps:

Great! You have improved your baseline model using polynomial terms, interaction terms, and regularizations, and have found your best model.

Now it's time to build an interactive dashboard to provide more interactive user-interactions.

Authors

Yan Luo

Other Contributors

Jeff Grossman

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-04-08	1.0	Yan	Initial version created