Regression Modeling G (6557)

Final Project

Semester 2 2021

Summary

1 Multiple Linear Regression

What it does

Multiple linear model expresses linear relationship between response and a vector of predictors. In the vector space of these predictors, it essentially finds a regression plane to best fit the data points as opposed to a straight line in a 2-D simple linear model. There will be a hyperplane for higher dimensional space that could not be rendered.

How it works

Same as simple linear model, it use ordinary least squares method, which minimizes the sum of square errors, to find the optimal coefficients. And then apply f-test via ANOVA table to evaluate the model.

The pros & cons

- Pros:
  - It is fast regardless of the size of dataset.
  - Intuitive interpretation of the coefficient.
- Cons:
  - Susceptible to outliers.
  - Collinearity can affect the performance dramatically.

2 Elastic Net

What it does

In standard multiple linear regression, when there is additional relationship among the predictors, the coefficients may not work as it should be because of collinearity. Or there are too many predictors, which may lead to overfitting. To address those problems, regularization parameter is introduced. Predictors with little contribution to the final prediction performance will be penalized.

How it works

For ridge regression, a shrink term (squared bias, L2-norm) is used to reduce the impact of corresponding coefficients by making it close to zero. While for lasso regression, those coefficients are shrunk to zero with a term (absolute bias, L1-norm). Elastic Net basically incorporates both terms above, so coefficients with minor impact will be shrunk and the irrelevant ones will be set to zero.

The pros & cons

- Pros:
  - It combine the pros of both ridge and lasso. L1-norm effectively performs feature selection. L2-norm can stabilizes the progress of L1-norm and eliminates the limit of feature to be selected by it.
  - It encourages group effect if coefficients are highly correlated as opposed to lasso method which just simply put some of them to zero so less information will be removed.
- Cons:
  - Balance of L1-norm and L2-norm need to be tuned (lambda).

3 KNN

What it does

K nearest neighbours algorithm predict the response by comparing the input data points with the k most similar samples in the dataset. Usually, they are compared via euclidean distance (or other distance function) in the vector space so ther are called neighbours. The input will be given a prediction determined only by the k neighbours.

How it works

The response can be numerical or categorical so that is able to perform regression or classification job. The major parameter of the algorithm is k, which determines how many nearest samples are taken into consideration and then determines the performance. Optimal k is chosen by minimizing error metric such as RMSE for regression or maximizing accuracy of classification.

The pros & cons

- Pros:
  - Easy to understand and implement.
  - Robust to outliers.
  - Less requirement for input.
- Cons:
  - Susceptible to imbalanced dataset.
  - Computational expensive for high dimension and long data.
  - Hard to interpret underlying implication.

4 Poisson GLM

What it does

Generalized linear models (GLMs) extend the multiple linear regression model with other underlying probability distributions of response. Essentially, it allows us to deal with qualitative responses.

"Poisson" distribution is a discrete probability distribution which models count of a event in certain time frame, such as daily visitor to a restaurant. To inspect response of this kind with regard to predictors, we can use poisson GLM.

How it works

The poisson GLM expresses relationship between natural logarithmic value of response and a vector of predictors as a linear function. Predictors can be continuous or categorical. The "log" link function make sure the response is positive because we are using "poisson" which deal with count (positive integers).

Then we will apply maximum likelihood method to find the optimal coefficients, which in general maximizes the log-likehood of the model.

In terms of the coefficients, when the corresponding predictor increase by 1 unit (constant for others), the "log" response will increase by the value of this coefficient, or the response will be multiplied by exponential of the coefficient.

The pros & cons

- Pros:
  - No specific requirement for predictor.
  - Relatively easy and clear interpretation.
- Cons:
  - Perform poorly on "zero" count events.
  - Restrictive assumption on mean and variance.

Comparison

In general, the choice of model depends on application and dataset available. When normal distribution and independence assumption satisfied and strong linear relation existed, standard linear model can perform well enough. If regularization is needed to address overfitting, collinearity and so on, elastic net is considered. KNN is universal in many cases without strict assumption on distribution but can be computational expensive to implement. Poisson GLM works well with response which satisfied poisson distribution so it can be limited.

Case Study

Dataset: Seoul Bike Sharing Demand

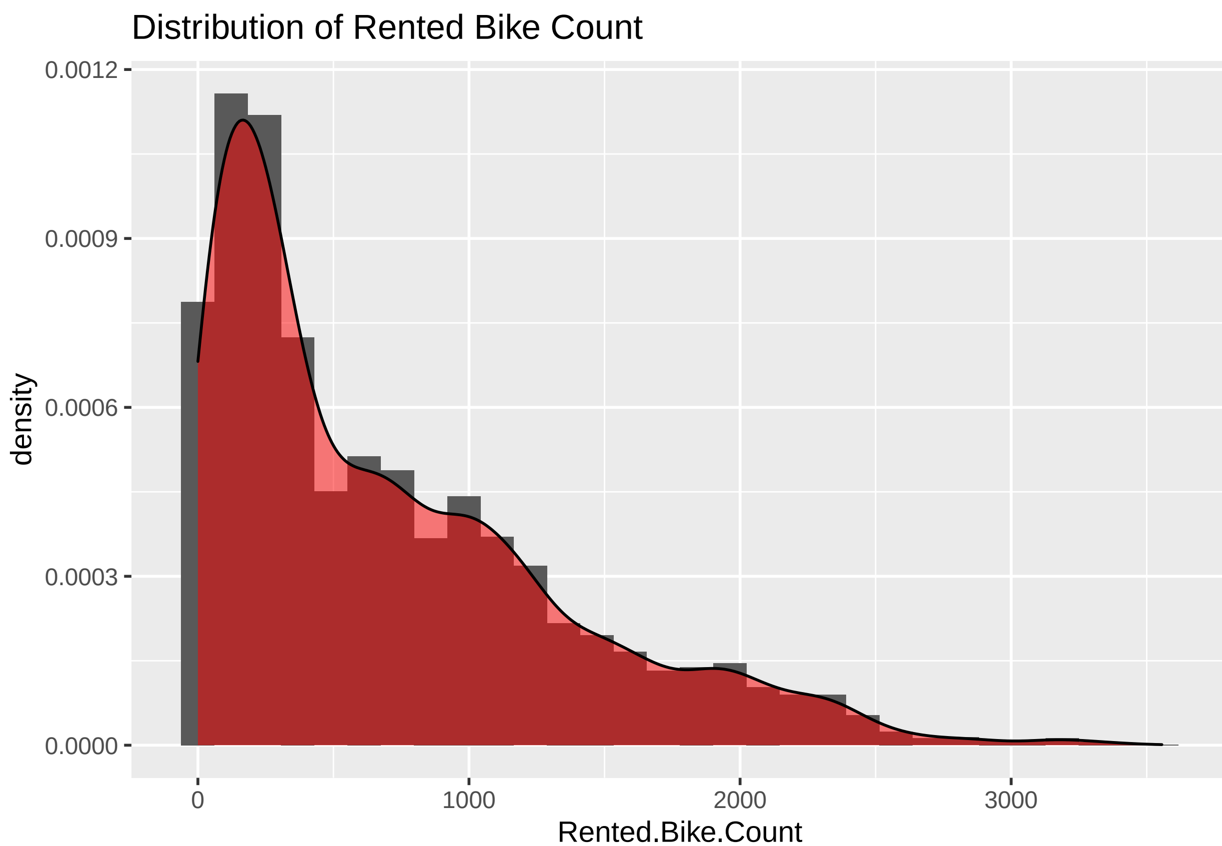
- Attribute Information:
  - Date : year-month-day
  - Rented Bike count - Count of bikes rented at each hour
  - Hour - Hour of he day
  - Temperature-Temperature in Celsius

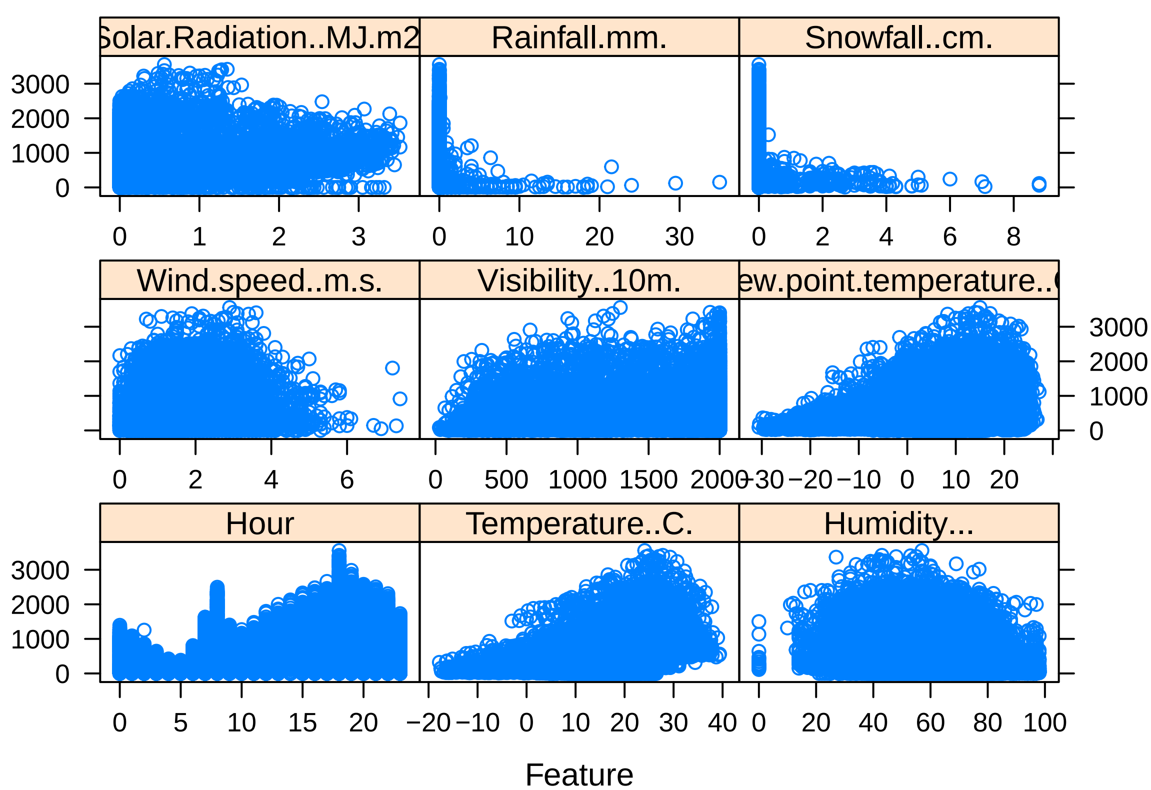
- Humidity - %
- Windspeed - m/s
- Visibility - 10m
- Dew point temperature - Celsius
- Solar radiation - MJ/m2
- Rainfall - mm
- Snowfall - cm
- Seasons - Winter, Spring, Summer, Autumn
- Holiday - Holiday/No holiday
- Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours)

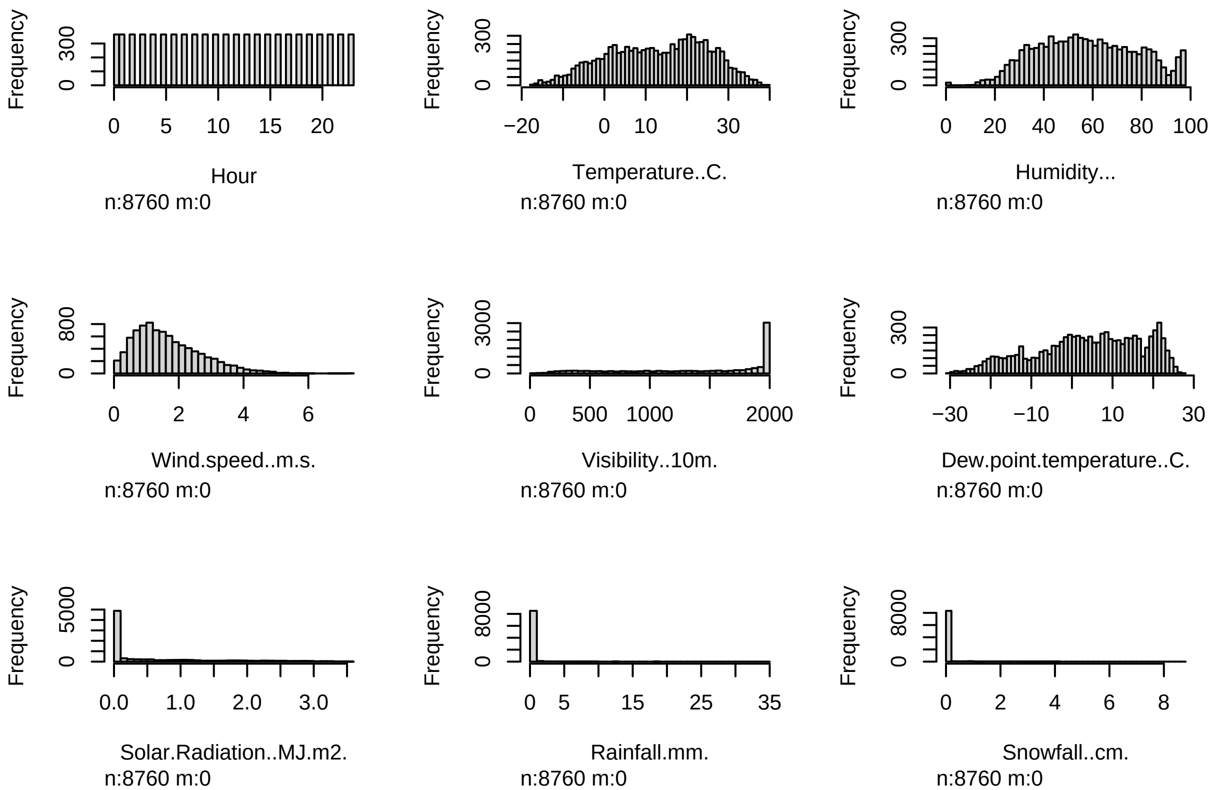
The objective is to predict the bike count required at each hour.

Check the dataset loaded.

Data Exploratory and Training







From the plot we can see, the response (Bike Count) can probably model by poisson distribution.

Multiple linear model

```r
# Multiple linear model.
modelLm <- train(
  Rented.Bike.Count ~ .,
  data = dataTrain,
  method = 'lm',
  trControl = trainControl('cv', number = 5))

summary(modelLm)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1185.65  -276.59   -57.95   210.99  2301.70
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   522.07979  100.34525   5.203 2.02e-07 ***
## Hour                           27.54096    0.83002  33.181  < 2e-16 ***
```

```
## Temperature..C.            19.18739    3.99032   4.808 1.55e-06 ***
## Humidity...                -9.73523    1.11344  -8.743  < 2e-16 ***
## Wind.speed..m.s.           18.96876    5.70936   3.322 0.000897 ***
## Visibility..10m.            0.01523    0.01113   1.368 0.171464
## Dew.point.temperature..C.   8.48338    4.16459   2.037 0.041685 *
## Solar.Radiation..MJ.m2.   -77.36448    8.60285  -8.993  < 2e-16 ***
## Rainfall.mm.              -57.23804    4.56929 -12.527  < 2e-16 ***
## Snowfall..cm.              32.14252   12.77055   2.517 0.011861 *
## SeasonsSpring             220.05920   20.86801  10.545  < 2e-16 ***
## SeasonsSummer             170.23017   31.63215   5.382 7.62e-08 ***
## SeasonsAutumn             349.06135   22.14851  15.760  < 2e-16 ***
## HolidayHoliday           -130.33811   24.18607  -5.389 7.32e-08 ***
## Functioning.DayNo        -912.24328   30.43325 -29.975  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 436.8 on 6993 degrees of freedom
## Multiple R-squared:  0.5444, Adjusted R-squared:  0.5435
## F-statistic: 596.8 on 14 and 6993 DF,  p-value: < 2.2e-16
```

We can see the "Visibility", "Dew point" and "Snowfall" are less relevant so will be removed.

```r
# Retrain
dataTrain <- dataTrain %>% select(-c(6, 7, 10))

start <- proc.time()

modelLm <- train(
  Rented.Bike.Count ~ .,
  data = dataTrain,
  method = 'lm',
  trControl = trainControl('cv', number = 5))

timerLm <- (proc.time() - start)['elapsed']

summary(modelLm)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1176.3  -279.8   -59.0   215.2  2300.3
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              372.2530    27.0846  13.744  < 2e-16 ***
## Hour                      27.4771     0.8274  33.207  < 2e-16 ***
## Temperature..C.           26.8597     0.9720  27.633  < 2e-16 ***
## Humidity...               -7.8051     0.3344 -23.340  < 2e-16 ***
## Wind.speed..m.s.          19.1434     5.6876   3.366 0.000767 ***
## Solar.Radiation..MJ.m2.  -81.8630     8.1415 -10.055  < 2e-16 ***
```

```
## Rainfall.mm.              -58.7699     4.5409 -12.942  < 2e-16 ***
## SeasonsSpring             214.2545    20.7013  10.350  < 2e-16 ***
## SeasonsSummer             176.7816    31.1435   5.676 1.43e-08 ***
## SeasonsAutumn             351.0939    21.7569  16.137  < 2e-16 ***
## HolidayHoliday           -130.7213    24.1817  -5.406 6.66e-08 ***
## Functioning.DayNo        -910.9103    30.4287 -29.936  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 437 on 6996 degrees of freedom
## Multiple R-squared:  0.5436, Adjusted R-squared:  0.5429
## F-statistic: 757.5 on 11 and 6996 DF,  p-value: < 2.2e-16
```

```r
cat('Training time:', timerLm)
```

```
## Training time: 0.47
```

Then we have the linear model above.

```r
# Elastic Net.
start <- proc.time()

modelEn <- train(
  Rented.Bike.Count ~ .,
  data = dataTrain,
  method = 'glmnet',
  trControl = trainControl('cv', number = 5))

timerEn <- (proc.time() - start)['elapsed']

print(modelEn)
```

```
## glmnet
##
## 7008 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 5606, 5606, 5607, 5605, 5608
## Resampling results across tuning parameters:
##
##   alpha  lambda      RMSE      Rsquared   MAE
##   0.10    0.6949156  437.2501  0.5424989  327.2699
##   0.10    6.9491556  437.3120  0.5424398  326.9130
##   0.10   69.4915563  443.2109  0.5368495  326.8669
##   0.55    0.6949156  437.2528  0.5424853  327.2839
##   0.55    6.9491556  437.7864  0.5416807  326.2876
##   0.55   69.4915563  458.0811  0.5171696  335.5520
##   1.00    0.6949156  437.2586  0.5424735  327.2402
##   1.00    6.9491556  438.8447  0.5397846  326.1851
##   1.00   69.4915563  476.4802  0.4893006  351.2491
##
```

```
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.6949156.
```

```r
cat('Training time:', timerEn)
```

```
## Training time: 0.64
```

```r
# KNN.
start <- proc.time()

modelKnn <- train(
  Rented.Bike.Count ~ .,
  data = dataTrain,
  method = 'knn',
  trControl = trainControl('cv', number = 5))

timerKnn <- (proc.time() - start)['elapsed']

print(modelKnn)
```

```
## k-Nearest Neighbors
##
## 7008 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 5606, 5608, 5606, 5606, 5606
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared   MAE
##   5  358.5264  0.6962113  225.8167
##   7  356.2594  0.6982741  225.3669
##   9  356.2873  0.6976381  225.9481
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

```r
cat('Training time:', timerKnn)
```

```
## Training time: 1.47
```

```r
# Poisson GLM.
start <- proc.time()

modelPg <- train(
  Rented.Bike.Count ~ .,
  data = dataTrain,
  method = 'glmnet',
  family = "poisson",
  trControl = trainControl('cv', number = 5))
```

```r
timerPg <- (proc.time() - start)['elapsed']

print(modelPg)
```

```
## glmnet
##
## 7008 samples
##    9 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 5606, 5607, 5607, 5606, 5606
## Resampling results across tuning parameters:
##
##   alpha  lambda      RMSE      Rsquared   MAE
##   0.10    0.6949156  950.5601  0.3385469  697.8691
##   0.10    6.9491556  950.5614  0.4211477  697.9073
##   0.10   69.4915563  418.8351  0.5887672  293.6691
##   0.55    0.6949156  950.5601  0.3390599  697.8679
##   0.55    6.9491556  950.5623  0.4338451  697.9012
##   0.55   69.4915563  443.5982  0.5482993  320.6026
##   1.00    0.6949156  950.5602  0.3396253  697.8667
##   1.00    6.9491556  950.5637  0.4470472  697.8954
##   1.00   69.4915563  468.2164  0.5066154  346.9513
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.1 and lambda = 69.49156.
```

```r
cat('Training time:', timerPg)
```

```
## Training time: 3.8
```

Predict and Evaluate

```r
# Multiple linear model.
start <- proc.time()

predLm <- predict(modelLm, dataTest)

timerLm['predict'] <- (proc.time() - start)['elapsed']

metricLm <- postResample(predLm, dataTest$Rented.Bike.Count) %>% print()
```

```
##        RMSE    Rsquared         MAE
## 418.1271569   0.5741188 311.6236175
```

```r
# Elastic Net.
start <- proc.time()
```

```r
predEn <- predict(modelEn, dataTest)

timerEn['predict'] <- (proc.time() - start)['elapsed']

metricEn <- postResample(predEn, dataTest$Rented.Bike.Count) %>% print()
```

```
##        RMSE    Rsquared         MAE
## 418.3175528   0.5740348 311.5743315
```

```r
# KNN.
start <- proc.time()

predKnn <- predict(modelKnn, dataTest)

timerKnn['predict'] <- (proc.time() - start)['elapsed']

metricKnn <- postResample(predKnn, dataTest$Rented.Bike.Count) %>% print()
```

```
##        RMSE    Rsquared         MAE
## 350.3603594   0.7062959 218.0678205
```

```r
# Poisson GLM.
start <- proc.time()

predPg <- predict(modelPg, dataTest)

timerPg['predict'] <- (proc.time() - start)['elapsed']

metricPg <- postResample(predPg, dataTest$Rented.Bike.Count) %>% print()
```

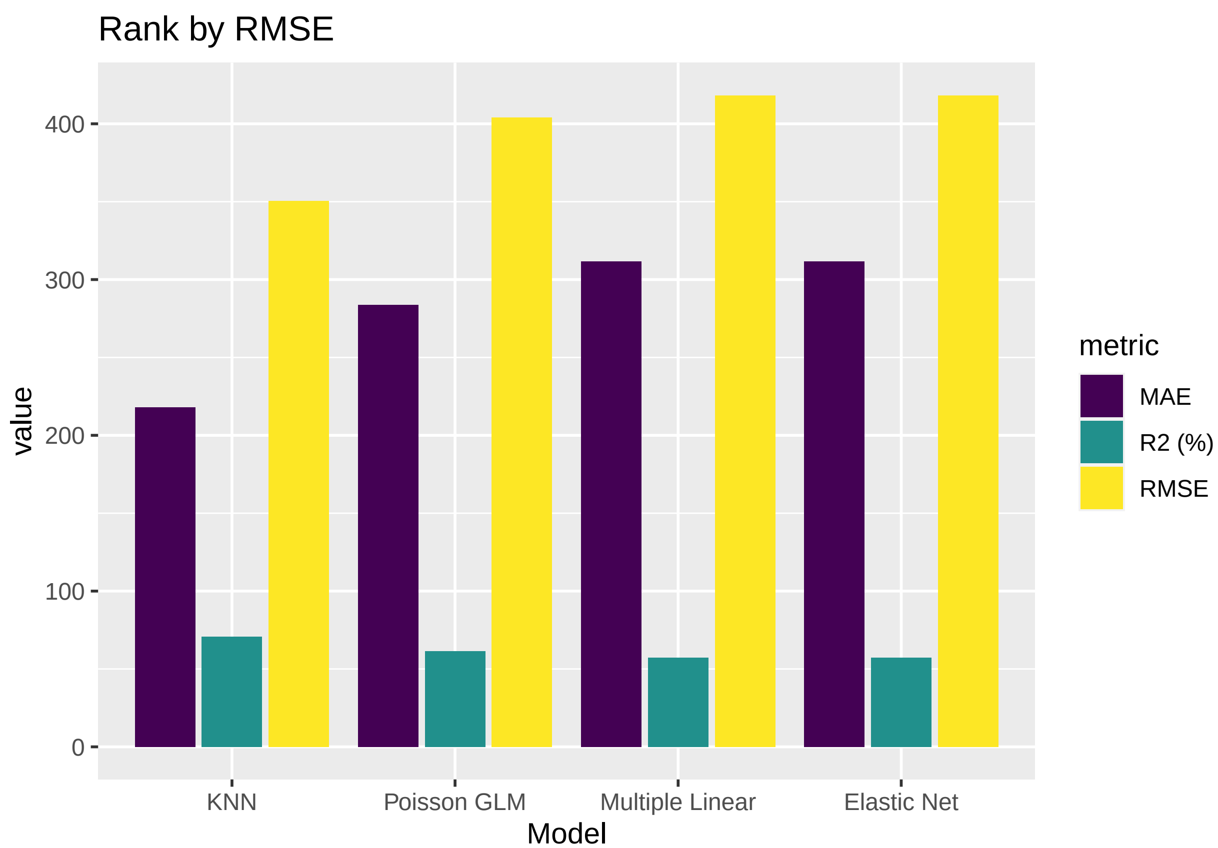
```
##        RMSE    Rsquared         MAE
## 403.9806044   0.6139976 283.8915289
```

Compare models

```
## # A tibble: 4 x 6
##   Model              RMSE Rsquared   MAE Train Predict
##   <chr>             <dbl>    <dbl> <dbl> <dbl>   <dbl>
## 1 Multiple Linear  418.     0.574  312. 0.470   0
## 2 Elastic Net      418.     0.574  312. 0.640   0
## 3 KNN              350.     0.706  218. 1.47    0.110
## 4 Poisson GLM      404.     0.614  284. 3.8     0
```

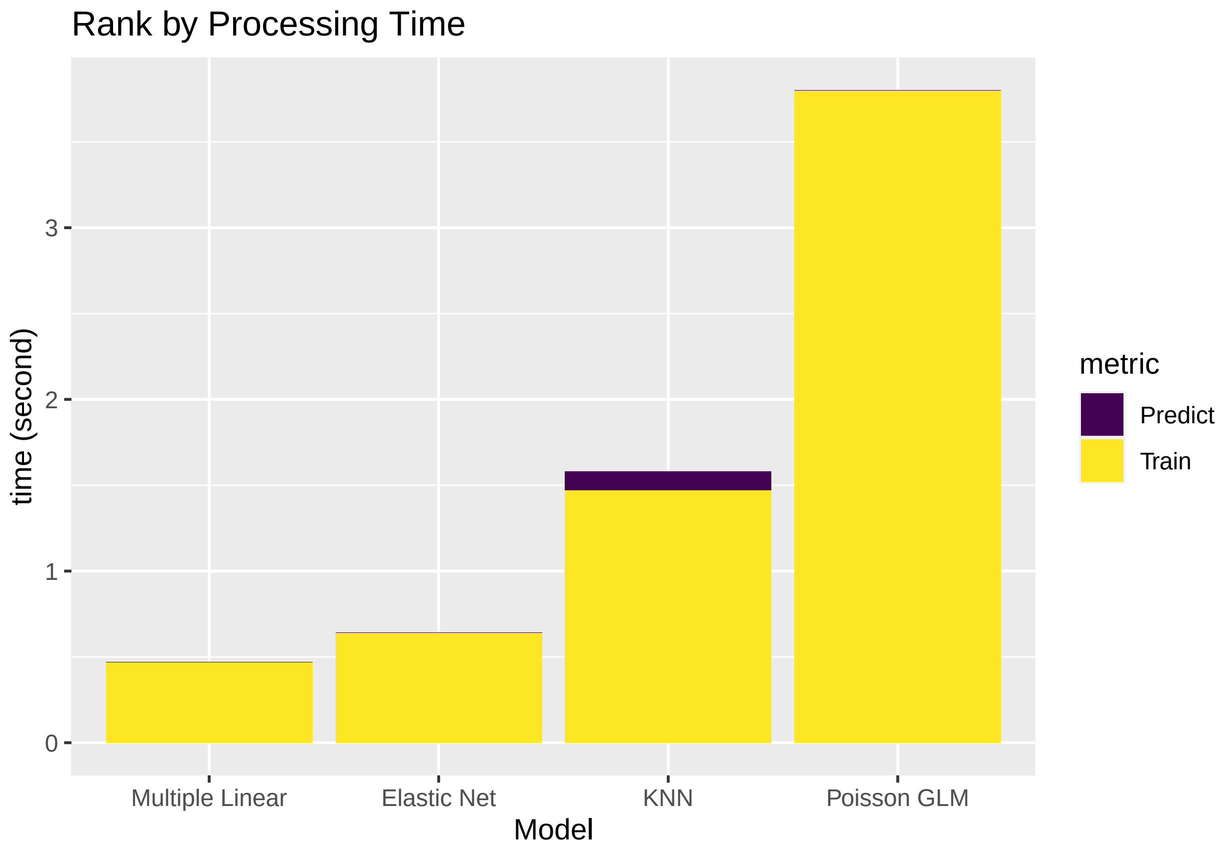
Apparently, KNN works better in this case.

Rank by RMSE



As we can see from the ranking, KNN shows best holistic performance, followed by Poisson GLM. KNN's better performance may due to its resistant to noisy data in this case. Poisson GLM benefit from that fact that the response follows poisson well.

Rank by Processing time



While KNN and Poisson GLM shows better prediction performance, they did take longer time to train. In the meantime, KNN requires longer time to do prediction which is the natural of the algorithm. Because it still needs to find and do calculation on the k neighbours in each prediction process. Its training is to find the optimal k without a regression function like the others.

References

- Sathishkumar V E, Jangwoo Park, and Yongyun Cho. 'Using data mining techniques for bike sharing demand prediction in metropolitan city.' Computer Communications, Vol.153, pp.353-366, March, 2020
- Sathishkumar V E and Yongyun Cho. 'A rule-based model for Seoul Bike sharing demand prediction using weather data' European Journal of Remote Sensing, pp. 1-18, Feb, 2020
- Applied Regression Modeling Third Edition by Iain Pardoe