Gas Sensor Array Harvard Capstone Project

Michael Hauffe

2025-08-07

##Introduction/Overview/Executive Summary ############################## The goal of this project is to compare 4 different machine learning models for predicting Carbon Monoxide concentration in an Ethylene and Air mixture. The dataset used is called “Gas sensor array under dynamic gas mixtures” on Kaggle and was created by the University of California San Diego. The data was gathered by creating different concentration mixtures of Ethylene and Carbon Monoxide (also known as “CO”) with air and then measuring the resulting gas mixture with 16 sensors. Because Carbon Monoxide is a potential product of the combustion of Ethylene, methods for determining the concentration of CO in Ethylene and air will have real world applications. I will explore the data, prepare the data, fit it to 4 different models, then compare the RMSE (Root Mean Squared Error) results at the end.

##Methods and Analysis ##############################

I chose 4 popular and powerful models for this exercise: K Nearest Neighbors, Generalized Linear Model, Neural Network, and Bayesian Regularized Neural Network. Here is why I chose those specifically. All 4 are designed to work for regression tasks, all 4 are included in the caret package, and the first 3 are quite popular which makes them useful for model comparison. I’ve chosen not to tune any hyperparameters so as to compare these models as they come. I’m also curious to see if the Bayesian Regularized Neural Network fairs better than the standard neural network, so I’ve included that for comparison. In the end, I have 4 models in total.

```
#I set the seed to 500. The seed is a usually arbitrary number that
determines the random events of the code. I set it manually so that my code
runs the same way each time
set.seed(500)
#I select a portion of the data so that model run times are reasonable. I
randomize the selection so that I still get a representative sample of the
dataset.
df <- completedf[sample(nrow(completedf),50000),]

#This is where I explore the data
#First, I print the first 10 rows. This allows me to see that the header
assignment worked as expected and gives me an idea of the typical values in
```

```r
the dataset
print(head(df,10))
```

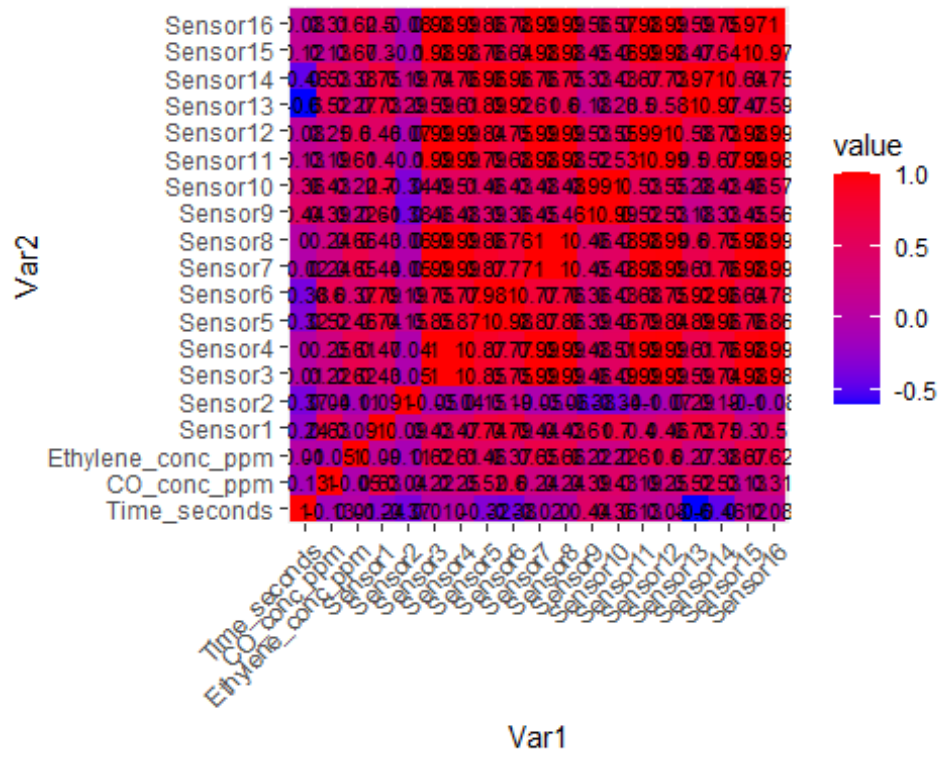
```
##         Time_seconds CO_conc_ppm Ethylene_conc_ppm Sensor1 Sensor2 Sensor3
## 3045301     30457.83      213.33              0.00 2598.68   -3.58 3675.21
## 2026732     20271.91        0.00              0.00 1068.48  151.18 1351.08
## 3061290     30617.72        0.00              0.00 2133.42   32.82 2693.69
## 3531922     35324.17      333.33             11.67 2973.25   16.96 6088.40
## 2465322     24657.78      293.33             11.67 3157.73   10.75 6190.15
## 3686624     36871.20      253.33             11.67 2756.95   20.39 4168.11
## 118576       1185.82      200.00              0.00 2384.35 4352.46 3425.99
## 1411638     14120.92        0.00              0.00 1342.91  183.48 4996.70
## 774246       7743.00      240.00              0.00 2840.93 1934.50 3946.84
## 3154278     31547.60        0.00              0.00 1016.68   19.73 1443.47
##         Sensor4 Sensor5 Sensor6 Sensor7 Sensor8 Sensor9 Sensor10 Sensor11
## 3045301 4093.68 1803.58 2182.74 4138.44 4392.47 1553.87  1552.56  4190.03
## 2026732 1506.91 1099.67 1469.32 1540.63 1678.61  794.29   737.13  1643.42
## 3061290 3029.92 1413.50 1761.33 2869.44 3005.38 1364.94  1390.22  3150.68
## 3531922 6734.76 2312.87 2591.71 7189.38 7633.38 1956.80  1937.75  7003.38
## 2465322 6814.91 2363.91 2662.30 7379.01 7819.03 1981.51  1915.52  6895.83
## 3686624 4668.39 1914.37 2278.91 4799.83 5143.82 1633.06  1642.76  4893.46
## 118576  3911.78 2217.38 2701.30 4012.04 4207.42  303.37   424.51  3435.97
## 1411638 5389.62 1746.94 1996.25 5406.99 5674.67  912.38   932.97  5578.49
## 774246  4417.12 2081.88 2473.10 4561.66 4828.40 1172.74  1373.35  4214.98
## 3154278 1635.04 1102.48 1465.90 1633.06 1753.23  898.83   884.35  1815.16
##         Sensor12 Sensor13 Sensor14 Sensor15 Sensor16
## 3045301  3458.93   884.55  1010.36  4177.59  3878.47
## 2026732  1339.56   678.90   734.88  1827.44  1371.04
## 3061290  2508.65   774.60   872.03  2942.64  2562.88
## 3531922  5658.69   990.47  1141.13  7548.77  6380.51
## 2465322  5600.24  1033.69  1209.84  7546.26  6436.16
## 3686624  3953.81   903.86  1012.53  4987.56  4481.69
## 118576   3023.02  1155.29  1158.53  3634.25  3311.38
## 1411638  4261.84   880.88   999.32  5846.90  4276.42
## 774246   3528.57  1061.50  1172.74  4291.03  3912.73
## 3154278  1419.65   652.06   720.46  1895.68  1485.62
```

```r
#I print the structure in this line. It tells me about the shape of the datset, if there's any nested values (which there aren't in this case), the data types of the columns, and the names of the columns. Knowing this information is very helpful later on as I work with this data. For example, if the Time_seconds column were a string instead of a number, that would tell me something went wrong with the data import and prompt me to investigate further. I also notice that many of the columns have a lot of overlapping data. For example, CO_conc_ppm has a Min of 0, a 1st Quartile of 0, and a Median that's also 0. I see other columns that also have a lot of repeating or close values. These values appear to be close in feature space.
print(str(df))
```

```
## 'data.frame':    50000 obs. of  19 variables:
##  $ Time_seconds     : num  30458 20272 30618 35324 24658 ...
##  $ CO_conc_ppm      : num  213 0 0 333 293 ...
##  $ Ethylene_conc_ppm: num  0 0 0 11.7 11.7 ...
##  $ Sensor1          : num  2599 1068 2133 2973 3158 ...
##  $ Sensor2          : num  -3.58 151.18 32.82 16.96 10.75 ...
##  $ Sensor3          : num  3675 1351 2694 6088 6190 ...
##  $ Sensor4          : num  4094 1507 3030 6735 6815 ...
##  $ Sensor5          : num  1804 1100 1414 2313 2364 ...
##  $ Sensor6          : num  2183 1469 1761 2592 2662 ...
##  $ Sensor7          : num  4138 1541 2869 7189 7379 ...
##  $ Sensor8          : num  4392 1679 3005 7633 7819 ...
##  $ Sensor9          : num  1554 794 1365 1957 1982 ...
##  $ Sensor10         : num  1553 737 1390 1938 1916 ...
##  $ Sensor11         : num  4190 1643 3151 7003 6896 ...
##  $ Sensor12         : num  3459 1340 2509 5659 5600 ...
##  $ Sensor13         : num  885 679 775 990 1034 ...
##  $ Sensor14         : num  1010 735 872 1141 1210 ...
##  $ Sensor15         : num  4178 1827 2943 7549 7546 ...
##  $ Sensor16         : num  3878 1371 2563 6381 6436 ...
## NULL

#This line tells me there are no null values in the dataset. This is one data
cleaning step I no longer have to do
print(colSums(is.na(df)))

##      Time_seconds       CO_conc_ppm Ethylene_conc_ppm           Sensor1
##                 0                 0                 0                 0
##           Sensor2           Sensor3           Sensor4           Sensor5
##                 0                 0                 0                 0
##           Sensor6           Sensor7           Sensor8           Sensor9
##                 0                 0                 0                 0
##          Sensor10          Sensor11          Sensor12          Sensor13
##                 0                 0                 0                 0
##          Sensor14          Sensor15          Sensor16
##                 0                 0                 0

#Finally, the summary gives me some statistics about the distribution of the
data. Because of this line, I now know CO_conc_ppm ranges from 0-533.3. This
can help inform me later on concerning whether a particular RMSE is good or
bad
print(summary(df))

##   Time_seconds       CO_conc_ppm    Ethylene_conc_ppm    Sensor1
##  Min.   :    0.92   Min.   :  0.0   Min.   : 0.000    Min.   : -45.2
##  1st Qu.:10597.39   1st Qu.:  0.0   1st Qu.: 0.000    1st Qu.:1323.2
##  Median :21120.07   Median :  0.0   Median : 6.670    Median :1785.2
##  Mean   :21073.12   Mean   :127.2   Mean   : 5.317    Mean   :2061.9
##  3rd Qu.:31625.74   3rd Qu.:266.7   3rd Qu.:10.000    3rd Qu.:2865.5
##  Max.   :42087.44   Max.   :533.3   Max.   :20.000    Max.   :3544.4
##     Sensor2           Sensor3          Sensor4           Sensor5
```

```
##  Min.   :  -31.64   Min.   : -31.8   Min.   :   2.44   Min.   :  -0.98
##  1st Qu.:    8.96   1st Qu.:2948.1   1st Qu.:3239.10   1st Qu.:1497.44
##  Median :   39.38   Median :4494.7   Median :4948.54   Median :1983.27
##  Mean   :  508.59   Mean   :4365.3   Mean   :4791.62   Mean   :1893.14
##  3rd Qu.:  159.57   3rd Qu.:5661.5   3rd Qu.:6176.08   3rd Qu.:2260.54
##  Max.   :27628.61   Max.   :8583.3   Max.   :9206.00   Max.   :2893.21
##     Sensor6           Sensor7           Sensor8           Sensor9
##  Min.   : -26.78   Min.   :  -8.94   Min.   :   -1.46   Min.   :  163.3
##  1st Qu.:1853.68   1st Qu.:3205.45   1st Qu.: 3415.43   1st Qu.:  933.5
##  Median :2271.56   Median :5228.84   Median : 5541.45   Median : 1138.9
##  Mean   :2212.92   Mean   :5078.11   Mean   : 5376.50   Mean   : 1184.0
##  3rd Qu.:2566.99   3rd Qu.:6717.46   3rd Qu.: 7101.82   3rd Qu.: 1539.1
##  Max.   :3498.25   Max.   :9859.71   Max.   :10478.63   Max.   :12314.2
##     Sensor10          Sensor11          Sensor12          Sensor13
##  Min.   :  215.7   Min.   : 884.7   Min.   : 895.2   Min.   : 616.9
##  1st Qu.:  941.1   1st Qu.:3409.8   1st Qu.:2696.0   1st Qu.: 807.0
##  Median : 1173.5   Median :5015.9   Median :4118.0   Median : 928.3
##  Mean   : 1216.7   Mean   :4863.4   Mean   :3919.6   Mean   : 927.3
##  3rd Qu.: 1554.1   3rd Qu.:6126.8   3rd Qu.:4967.5   3rd Qu.:1032.9
##  Max.   :13515.9   Max.   :9320.1   Max.   :7273.5   Max.   :7976.5
##     Sensor14          Sensor15         Sensor16
##  Min.   : 678.0   Min.   :  563   Min.   : 500.6
##  1st Qu.: 908.9   1st Qu.: 3609   1st Qu.:2775.2
##  Median :1059.7   Median : 5312   Median :4541.0
##  Mean   :1034.7   Mean   : 5281   Mean   :4302.7
##  3rd Qu.:1161.0   3rd Qu.: 6738   3rd Qu.:5562.3
##  Max.   :7307.0   Max.   :10638   Max.   :7853.7
```

```
#I have a lot of columns here and reducing the total number of them could
help my models run faster. To figure out which columns to remove, I make a
correlation matrix. This will tell me how linearly related different columns'
values are.

#Calculate the correlations
cor_matrix <- cor(df)
#Melt the data into the correct format
cor_melt <- melt(cor_matrix)
#Create the correlation matrix with the values made above.
ggplot(cor_melt, aes(Var1, Var2, fill = value)) + geom_tile() +
geom_text(aes(label=round(value,2)), color = "black", size = 3) +
theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1)) +
scale_fill_gradient(low="blue", high = "red")
```




```
#It looks like there are some columns that aren't correlated with CO_conc_ppm
nearly at all. To make the model run faster, I'll remove those columns from
the dataset

#Pull the correlations for CO_conc_ppm
correlation_vector <- cor(df)[, "CO_conc_ppm"]
#Create the list of columns with correlations greater than 0.15. 0.15 is an
arbitrary number that I picked. Had I picked a greater number, I would have
removed more columns. It's possible that these columns that I'm removing
would have been useful in at least one of the models created, however, due to
lack of correlation, I make the educated guess that these columns would be
less useful than the ones kept.
cols_to_keep <- names(correlation_vector)[abs(correlation_vector) >= 0.15]
#Removing the columns that are not in the cols_to_keep variable
df <- df[,cols_to_keep]



#The metric I will be using to grade each model's performance is RMSE.
Because I will be using it often, I will make a function for it.
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

#Creating my train and test set. 20% allocated to the test set is convention.
Because I have a large dataset, I could have also gotten away with using a
```

```r
little less towards testing if I wanted to.
test_index <- createDataPartition(y = df$`CO_conc_ppm`, times = 1,p = 0.2,
list = FALSE)
train_set <- df[-test_index,]
test_set <- df[test_index,]

#Printing the RMSE just using the mean of all values. This is to give me
something to compare to later on
BaselineRMSE <- RMSE(df$CO_conc_ppm, mean(df$CO_conc_ppm))
print(BaselineRMSE)
```

```
## [1] 161.0719
```

```r
#Fitting the KNN model
knn_fit <- train(CO_conc_ppm ~ ., data = train_set, method = "knn")
#Predicting using the KNN model
y_hat_knn <- predict(knn_fit, test_set)
#Printing the RMSE
KNNRMSE <-RMSE(test_set$CO_conc_ppm, y_hat_knn)
print(KNNRMSE)
```

```
## [1] 39.04079
```

```r
#Fitting the glm model
glm_fit <- train(CO_conc_ppm ~ ., data = train_set, method = "glm")
#Predicting using the glm model
y_hat_glm <- predict(glm_fit, test_set)
#Printing the RMSE
GLMRMSE <- RMSE(test_set$CO_conc_ppm, y_hat_glm)
print(GLMRMSE)
```

```
## [1] 97.29376
```

```r
#Fitting the Standard Neural Network model
nnet_fit <- train(CO_conc_ppm ~ ., data = train_set, method = "nnet")

#Predicting using the nnet model
y_hat_nnet <- predict(nnet_fit, test_set)
#printing the RMSE
NNETRMSE <- RMSE(test_set$CO_conc_ppm, y_hat_nnet)
print(NNETRMSE)
```

```
## [1] 203.4218
```

```r
#Fitting the Bayesian Regularized Neural Network model
brnn_fit <- train(CO_conc_ppm ~ ., data = train_set, method = "brnn")

#Predicting using the brnn model
y_hat_brnn <- predict(brnn_fit, test_set)
#printing the RMSE
BRNNRMSE <- RMSE(test_set$CO_conc_ppm, y_hat_brnn)
print(BRNNRMSE)
```

```
## [1] 80.50905
```

##Results ##############################

```
RMSEdf <- data.frame(Method = c("Predict Using Only The Mean","K Nearest
Neighbors","Generalized Linear Model","Neural Network","Bayesian Regularized
Neural Network"),
                     RMSE =
c(BaselineRMSE,KNNRMSE,GLMRMSE,NNETRMSE,BRNNRMSE))
print(RMSEdf %>% arrange(RMSE))
```

```
##                                Method      RMSE
## 1                 K Nearest Neighbors  39.04079
## 2 Bayesian Regularized Neural Network  80.50905
## 3            Generalized Linear Model  97.29376
## 4         Predict Using Only The Mean 161.07191
## 5                      Neural Network 203.42177
```

K Nearest Neighbors did the best by far with an RMSE of 39.04079. I suspect that the good performance is due to the fact that the data points had a lot of near overlapping data points. The values were close in feature space, leading to good performance on a model that draws predictions from the nearest points in feature space. The standard Neural Network performed the worst, despite the model's reputation for achieving amazing results. It even performed worse than predicting using only the mean. I suspect this is because I used the model as it comes without any tuning. It could be that Neural Networks are especially susceptible to incorrectly tuned hyperparameters. Since the introduction of Bayesian Regularization appears to have improved the model considerably, I suspect overfitting may also have a part in the standard Neural Network's poor performance.

##Conclusion ############################## I tried 4 different out of the box machine learning models to predict the Carbon Monoxide concentration in Ethylene and Air. Those models were K Nearest Neighbors, Generalized Linear Model, Neural Network, and Bayesian Regularized Neural Network. In the end, the K Nearest Neighbors did the best with an RMSE of 39.04079 and the standard Neural Network fared the worst with an RMSE of 203.42177. These findings could prove helpful in the chemical industry in the use of Carbon Monoxide concentration testing. A major limitation of these models is the lack of hyperparameter tuning. This tuning was omitted from this project for the sake of calculation speed and so that the base models could be compared. For future work, I would like to repeat this project, but with hyperparameter tuning. I suspect the neural networks especially would fare significantly better in that case.An ensemble of the models that perform better than just the mean could be made for even better model performance.

##References ############################## Ethylene combustion chemistry learned from my Chemistry Bachelors degree, so no reference is needed

https://www.kaggle.com/datasets/uciml/gas-sensor-array-under-dynamic-gas-mixtures/data