Tutorial #3

Machine learning algorithms can be divided into four groups:

1. Unsupervised Classification
2. Supervised Classification
3. Supervised Regression
4. Unsupervised Regression (Explain this)

Explain each of the four types in your own words. What are some of the examples of each type? State some applications of each

For STLF, we are interested in Supervised Regression. In tutorial 2, we discussed linear regression (A supervised regression algorithm), in which we devise the equation of the model. However, in more complex machine learning models, we do not have to build the whole model from scratch. These models work like black box. We can only control hyper parameters which provide us some control over the response of the model.

One such model is SVM (Support Vector Machine). Read about this here. SVM was originally developed as a Supervised Classification Algorithm. However, SVR (Support Vector Regression), a variant of SVM, is used for regression.

Mileage

One of the most important feature of an automobile is the distance that it can travel in one gallon of fuel. Mileage per gallon (mpg) can be used to compare different vehicles. However, to create a comparison between various vehicles it is important that **mpg** for each vehicle is measured under exactly identical conditions which is thoretically impossible. So we need some indirect method to calculate **mpg** using other attributes of a vehicle such as number of cylinders, horsepower and weight.

For this purpose we will use Auto MPG Dataset. The dataset contains 9 attributes of 398 vehicles. These attributes are as follows:

1. mpg: Miles per gallon of the engine
2. cylinders: number of cylinders in the engine
3. displacement: engine displacement
4. horsepower: horsepower of the car
5. weight: weight of the car (lbs)
6. acceleration: acceleration of the car. Most likely (s) it took the car from 0-60
7. model year: model year of the car in the 1900s
8. origin: unknown descriptor
9. car name: Name of the car

The code below loads the dataset and also shows the structure of each column of the dataframe.

```
# Loading the dataset auto_mpg =
read.csv("auto-mpg.csv")

# Structure of the dataset str(auto_mpg)
```

```
## 'data.frame':                 398 obs. of 9 variables:
## $ mpg                        : num 18 15 18 16 17 15 14 14 14 15 ...
## $ cylinders                   : int 8 8 8 8 8 8 8 8 8 8 ...
## $ displacement: num 307 350 318 304 302 429 454 440 455 390 ...
## $ horsepower : Factor w/ 94 levels "?","100","102",..: 17 35 29 29 24 42 47 46 48 40 ...
## $ weight                          : int 3504 3693 3436 3433 3449 4341 4354 4312 4425 3850 ...
## $ acceleration: num 12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
## $ model.year : int 70 70 70 70 70 70 70 70 70 70 ...
## $ origin                          : int 1 1 1 1 1 1 1 1 1 1 ...
## $ car.name : Factor w/ 305 levels "amc ambassador brougham",..: 50 37 232 15 162 142 55 224 242 2
```
I expected that the **horsepower** column will be numeric or integer same as **mpg** and **cylinders**. However, it's data type is Factor. This is because it not only conatins numeric values but also a special character i.e. "?". We need to remove the rows with "?" in **horsepower** column. The following code will do this.

```
# Rows with "?" in horsepower column are
removed auto_mpg =
auto_mpg[auto_mpg$horsepower != "?",]


# converting horsepower column to numeric
auto_mpg$horsepower =
as.numeric(auto_mpg$horsepower)
```

Next, we don't know what **origin** column means, so removing it.

```
# Origin is the 8th column
auto_mpg = auto_mpg[,-
8]
```

The following code shows the summary of all the columns in the dataset. Most of the cars in the dataset are from 1970s. The average weight of the automobile is 2978 lbs.

```
summary(auto_mpg)
```

```
##            mpg           cylinders        displacement      horsepower
## Min.         : 9.00  Min.        :3.000  Min.      : 68.0  Min.      : 2.00
## 1st Qu.:17.00        1st Qu.:4.000       1st Qu.:105.0     1st Qu.:26.75
## Median :22.75        Median :4.000       Median :151.0     Median :62.00
## Mean         :23.45  Mean        :5.472  Mean      :194.4  Mean      :52.16
## 3rd Qu.:29.00        3rd Qu.:8.000       3rd Qu.:275.8     3rd Qu.:80.00
## Max.         :46.60  Max.        :8.000  Max.      :455.0  Max.      :94.00
##
##          weight      acceleration       model.year                 car.name
## Min.       :1613  Min.       : 8.00  Min.     :70.00  amc matador        : 5
## 1st Qu.:2225      1st Qu.:13.78      1st Qu.:73.00    ford pinto         : 5
## Median :2804      Median :15.50      Median :76.00    toyota corolla     : 5
## Mean       :2978  Mean       :15.54  Mean     :75.98  amc gremlin        : 4
## 3rd Qu.:3615      3rd Qu.:17.02      3rd Qu.:79.00    amc hornet         : 4
## Max.       :5140  Max.       :24.80  Max.     :82.00  chevrolet chevette: 4
##                                                       (Other)            :365
```

We are interested in the following questions:

1. Which car has the highest **mpg**?

```
as.vector(auto_mpg$car.name[auto_mpg$mpg == max(auto_mpg$mpg)])
## [1] "mazda glc"
```

Answer the following with code:

2. Which car has the lowest **mpg**?
3. What is the **mpg** of the heaviest car?
4. Does the car with the highest **horsepower** also has the highest **acceleration**?
5. What is the displacement of **audi 5000s (diesel)**?

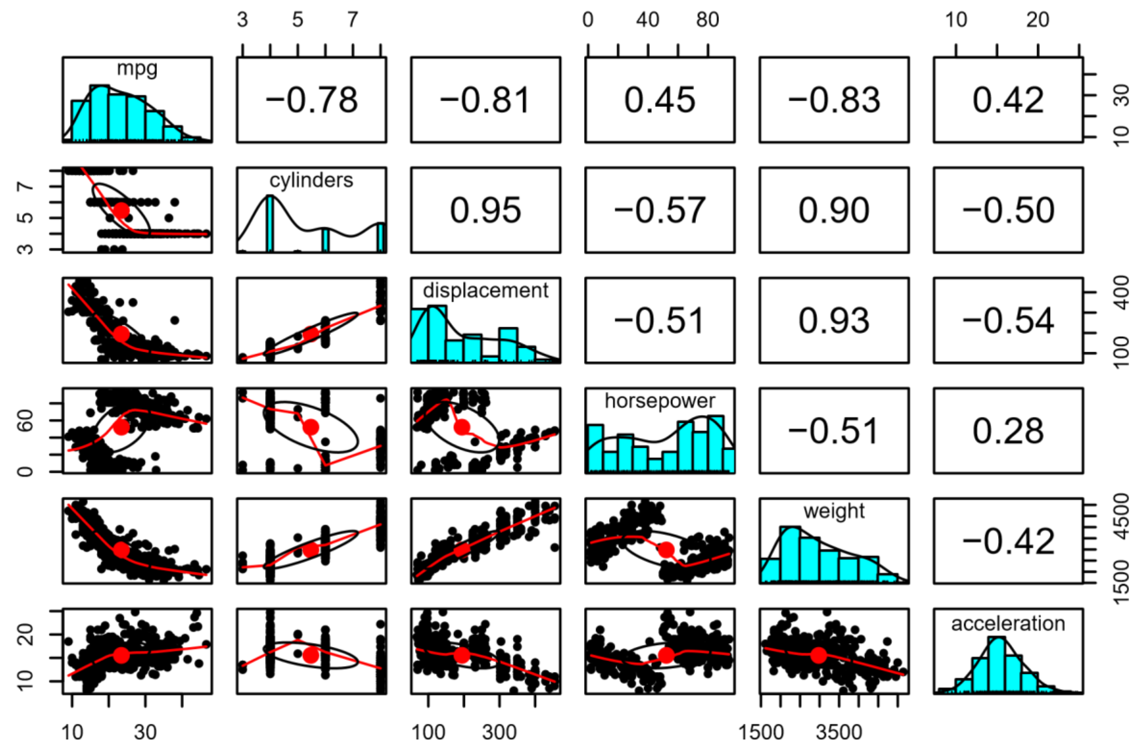
Now we need to understand the relation between the various attributes of the dataset. I have found this awesome function *pairs.panels()* in **psych** package for this purpose. **mpg** shows high negative correlation with **cylinders**, **displacement** and **weight**. *Is this expected? Explain.* **mpg** shows low but positive correlation with **horsepower** and **acceleration**.

If we use **weight** in our model, we do not have to use **cylinders** and **displacement**, as both have high positive correlation with **weight**. This is because of the phenomenon called collinearity.

What is the correlation between **horsepower** *and* **acceleration**?

```
# install.packages("psych") library(psych)

pairs.panels(auto_mpg[,-c(7,8)])
```



To predict the dependent variable, **mpg**, we will use three dependent variables, **horsepower**, **weight** and **acceleration**.

First we need to divide the dataset into train and test subsets.

```
smp_size <- floor(0.80 * nrow(auto_mpg))

train_ind <- sample(seq_len(nrow(auto_mpg)), size = smp_size) train_auto <-

auto_mpg[train_ind, ] test_auto <- auto_mpg[-train_ind, ]
```

We will use *e1071* package to implement SVR. The documentation of all *e1071* functions can be found here. *# install.packages("e1071")* **library**(e1071)

The code below is used to implement SVR.

- Although, we are implementing SVR, the function we are using is called *svm()*
- Writing the formula as *"mpg~horsepower+weight+acceleration"* means "predict **mpg** using **horsepower**, **weight** and **acceleration**". the independent variables are separated by "+". Read more about this here.
- Type is "eps-regression", which means the svm(), will fit a regression model and will try to decrease the error (epsilon) as much as possible. This is a hyper-parameter.
- Polynomial kernel is used. This means that a polynomial equation will be used as model equation. This is another hyper-parameter.
- The degree is 5 for this model. This is the degree of the polynomial equation.

```
# Run ?svm for help

svm_model = svm(mpg~horsepower+weight+acceleration, data =

train_auto, type = "eps-regression", kernel = "polynomial", degree = 5)

svm_model
```

```
##
## Call:
## svm(formula = mpg ~ horsepower + weight + acceleration, data = train_auto,
##                     type = "eps-regression", kernel = "polynomial", degree = 5)
## ##
## Parameters:
##            SVM-Type: eps-regression
## SVM-Kernel: polynomial
##                 cost: 1
##             degree: 5
##              gamma: 0.3333333
##             coef.0: 0
##            epsilon: 0.1
## ##
## Number of Support Vectors: 278
```

The output of the model shows several interesting parameters of the SVR fitted on the training data.

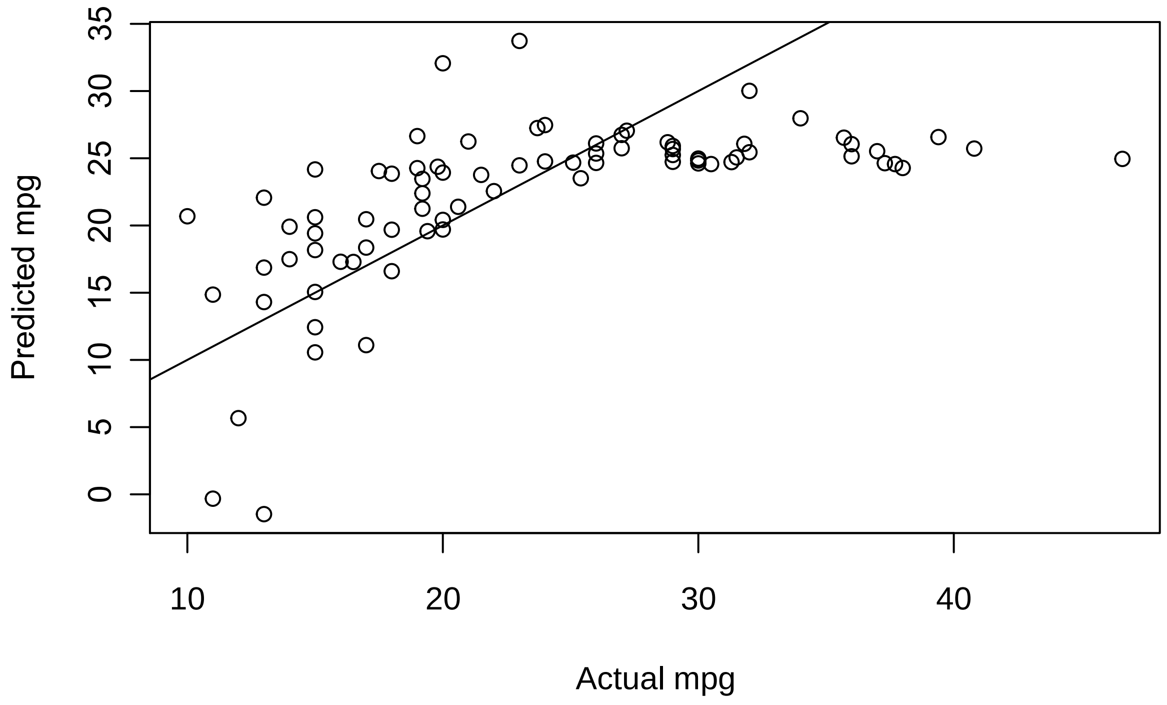
- *What is cost? Should it be 1 as it is in this model?*
- *What is gamma?*

The following code predicts the **mpg** is the test data and copies the result in **mpg_predicted** column in the *test_auto* datafranme.

```
test_auto$mpg_predicted = predict(svm_model, test_auto)
```

The plot below shows the actual vs predicted mpg from the test dataset. For a perfect model, we will expect all the data points aligned on the y=x line. However, as we can see some points are above and some are below the line, shows that the model is not accurate in its predictions.

```
plot(test_auto$mpg, test_auto$mpg_predicted, xlab = "Actual mpg", ylab = "Predicted mpg") abline(0,1)
```



MAE of 4.41 means that on average the model was off by 4.41 units in either direction in predicting **mpg**.

Interpret the other error metrics below.

In what case the MAPE and MPE are going to be too high and useless? Are they in this case?

```
library(forecast)

accuracy(test_auto$mpg, test_auto$mpg_predicted)
```

```
##                  ME     RMSE      MAE      MPE     MAPE
## Test set -1.277694 6.780702 5.179557 52.20773 79.03122
```

Create a new SVR model using **radial** *and* **sigmoid** *kernels. Note that both of these kernels do not use degree hyper-parameter.*

Which model is better? Why?

Concrete Slump Test

The concrete slump test measures the consistency of fresh concrete before it sets. It is performed to check the workability of freshly made concrete, and therefore the ease with which concrete flows.

This can be predicted using the concrete ingredients such as **cement** and **Slag**.

Use the Concrete Slump Test Data Set to build a SVR model to predict **SLUMP**. Use any set of hyperparameters and independent variables that you find logical.

The dataset is attached with the email.
Email me your detailed report.