INTRODUCTION TO MACHINE LEARNING
(NPFL054)
A template for Homework #2

Name: Milan Wikarski

School year: 2019/2020

- **Provide answers for the exercises.**
- **For each exercise, your answer cannot exceed one sheet of paper.**

1.1 Multiple linear regression

I have decided to perform a simple linear regression (one feature) for all numeric features of Auto dataset and calculate the correlation between these features and the target feature *mpg* because I was interested in differences between the models produced by simple linear regression and multiple linear regression (see out/simple-linear-regression.pdf):

I have performed multiple linear regression using all attributes except *name* by calling:

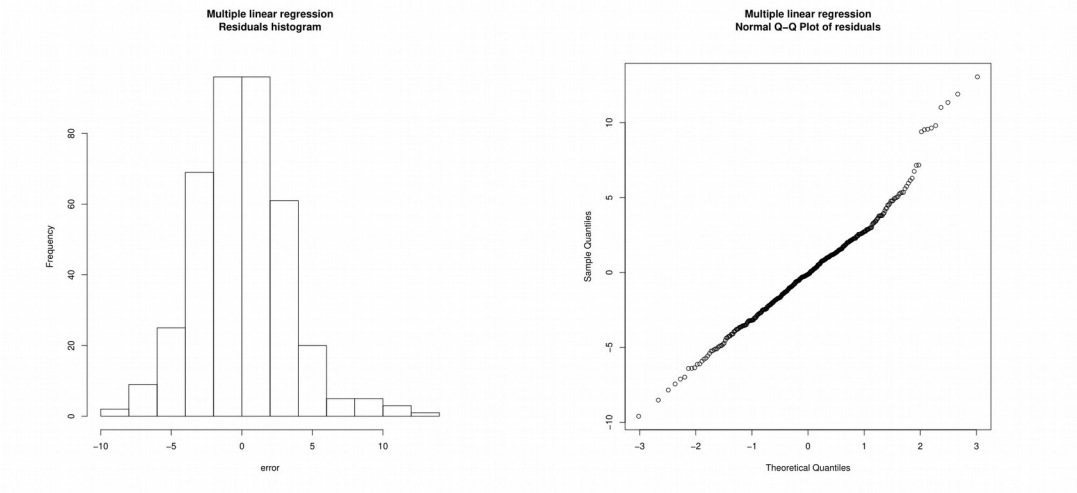
```
model <- lm(mpg ~ cylinders + displacement + horsepower + weight + acceleration
+ year + origin)
```

The outpus was:

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -17.218435   4.644294  -3.707  0.00024 ***
cylinders     -0.493376   0.323282  -1.526  0.12780
displacement   0.019896   0.007515   2.647  0.00844 **
horsepower    -0.016951   0.013787  -1.230  0.21963
weight        -0.006474   0.000652  -9.929  < 2e-16 ***
acceleration   0.080576   0.098845   0.815  0.41548
year           0.750773   0.050973  14.729  < 2e-16 ***
origin         1.426141   0.278136   5.127 4.67e-07 ***
---
Signif. Codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.328 on 384 degrees of freedom
Multiple R-squared:  0.8215,    Adjusted R-squared:  0.8182
F-statistic: 252.4 on 7 and 384 DF,  p-value: < 2.2e-16
```

We can see that *displacement, weight*, *year* and *origin* are significant in *mpg* prediction. Other features have t-value that is too high to be considered significant. This means that we cannot be sure if they really help us in any way to predict the target feature.

I have also explored the residuals, creating a historgram to see the distribution of the residual lengths and one Q-Q plot to see if they are normally distributed:



1.2 Polynomial regression

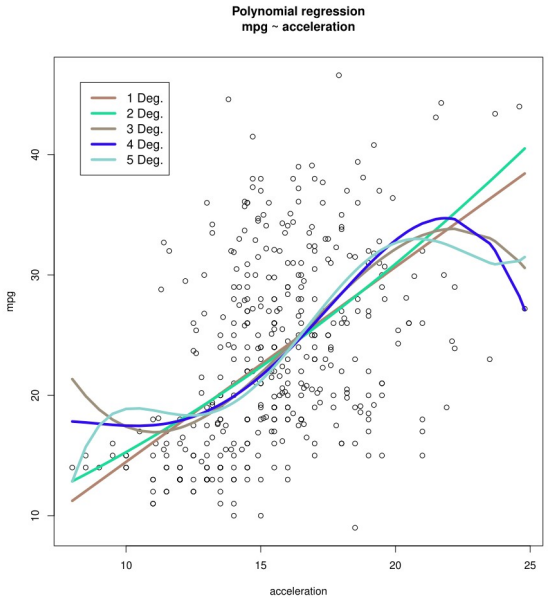
I have performed polynomial regression to predict *mpg* using *acceleration* by calling:

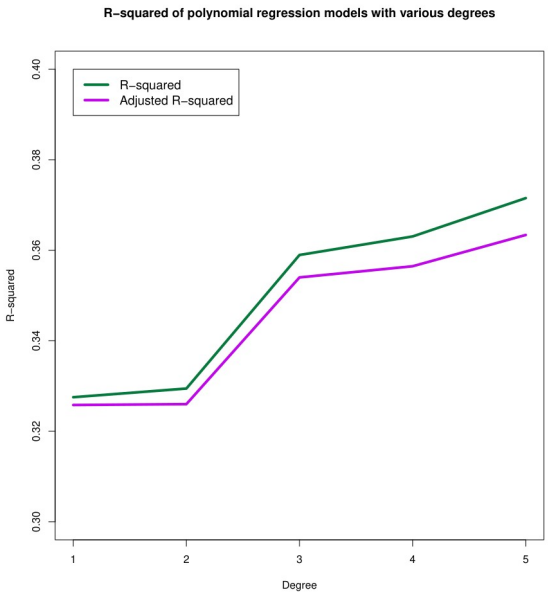
```
DEGREES_COUNT <- 5

for (degree in 1:DEGREES_COUNT) {
  # Create polynomial regression models degrees 1 to 5
  model <- lm(mpg ~ poly(acceleration.sorted, degree))

  [...]
}
```

saving associated R-squared valus in the loop. I have sorted the values of acceleration before performing polynomial regression. The values of R-squared and adjusted R-squared are:

```
> r.squared
[1] 0.3275221 0.3294382 0.3589554 0.3630519 0.3715209
> adj.r.squared
[1] 0.3257978 0.3259906 0.3539989 0.3564684 0.3633800
```

Afterwards, I have plotted two charts. One for the polynomial fits and one line chart for R-squared values:





2.1 Binary attribute `mpg01` and its entropy

First, I have extended the Auto dataset by *mpg01* feature calculated as follows:

```
data$mpg01 <- as.integer(data$mpg > median(data$mpg))
```

Then I have created data frame d containing all features including newly created *mpg01* and excluding the feature *mpg*:

```
d <- data[, c(2:10)]
```

Finally, I have calculated the entropy of the newly created *mpg01* feature using my function *entropy(x)*:

```
entropy(table(data$mpg01))
```

The output was:

```
> entropy(table(data$mpg01))
[1] 1
```

Although, this was not necessary, since *mpg01* is defined as 1 if above median; 0 if below median, the number of records in Auto dataset is even, hence the frequencies of 1 and 0 are going to be 50% / 50%, and therefore H(mpg01) = log_2(2) = 1

2.3 Trivial classifier accuracy

Trivial classifier will always predict the most frequent value, so the first step is to calculate the most frequent value. I have done it by calling:

```
trivial.predict <- as.integer(ifelse(table(train$mpg01)[1] > table(train$mpg01)
[2], 0, 1))
```

I have defined my own function *binary.evaluation(predictedValues, trueValues)* for binary classification evaluation. It returns a list of these attribtues:

- **confusion matrix**: 2x2 matrix where columns represent the prediction and rows represent the truth
- **true.positive**, **true.negative**, **false.positive**, **false.negative**: same values as in matrix, but names
- **precision**, **recall**, **specificity**, **F-score**, **accuracy**, **error**: model performance metrics

I have evaluated the trivial classifier by calling:

```
trivial.evaluation <- binary.evaluation(rep(trivial.predict, nrow(test)),
test$mpg01)
```

The accuracy of the trivial classifier is:

```
> trivial.evaluation$accuracy
[1] 0.4871795
```

2.4 Logistic regression – training and test error rate, confusion matrix, Sensitivity, Specificity, interpretation

---

I have performed logistic regression using all features except *name* and calculated the probabilities of each feature vector (where *{{set}}* is either *train* or *test*):

```
log <- glm(mpg01 ~ cylinders + displacement + horsepower + weight + acceleration
+ year + origin, data = train, family = binomial(link = "logit"))

log.probs.{{set}} <- predict(log, {{set}}, type = "response")
log.05.{{set}}.predict <- rep(0, length(log.probs.{{set}}))
log.05.{{set}}.predict[log.probs.{{set}} > 0.5] <- 1
```

Then, I have used my function to evaluate it on both train and test data set:

```
log.05.train.evaluation <- binary.evaluation(log.05.train.predict, train$mpg01)
log.05.test.evaluation <- binary.evaluation(log.05.test.predict, test$mpg01)
```

(a) The training error rate is:

```
> log.05.train.evaluation$error
[1] 0.08917197
```

(b) Confusion matrix, error rate, sensitivity (recall) and specificity evaluated on test set are:

```
$confusion.matrix
               Predicted positive Predicted negative
Truly positive                 34                  4
Truly negative                  5                 35

$precision     [1] 0.8717949
$recall        [1] 0.8947368
$specificity   [1] 0.875
$error         [1] 0.1153846
```

(c) The summary of logistic regression is:

```
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  -12.579718   6.375991  -1.973  0.04850 *
cylinders     -0.329955   0.473247  -0.697  0.48567
displacement   0.002390   0.013225   0.181  0.85659
horsepower    -0.057776   0.027141  -2.129  0.03328 *
weight        -0.003832   0.001260  -3.040  0.00236 **
acceleration  -0.078684   0.166533  -0.472  0.63658
year           0.406405   0.082565   4.922 8.55e-07 ***
origin         0.252230   0.403708   0.625  0.53211
---
Signif. Codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

We can see that the attribute *year* is the most significant in *mpg01* prediction. *Weight* is of less significance and *horsepower* of even lesser (but still significant). Other attributes can be neglected because of the high value of *Pr(>|z|)*.

2.5 Logistic regression – threshold 0.1 and 0.9, confusion matrix, Precision, Recall, F1-measure, interpretation

---

I have rerun the previous experiment using test data and two different thresholds for logistic regression – 0.1 and 0.9. I had already created the logistic regression model, so all I had to do was calculate the probabilities and evaluate. Here are the results:

Threshold 0.1

$confusion.matrix

```
               Predicted positive Predicted negative
Truly positive                 37                  1
Truly negative                  9                 31

$precision     [1] 0.8043478
$recall        [1] 0.9736842
$F.score       [1] 0.8809524
```

Threshold 0.9

```
$confusion.matrix
               Predicted positive Predicted negative
Truly positive                 23                 15
Truly negative                  0                 40

$precision     [1] 1
$recall        [1] 0.6052632
$F.score       [1] 0.7540984
```

I have decided that it would be interesting to explore this further and created a plot that shows how different performance metrics change as we tweek the threshold from 0 to 1 using the train data (because of the size). (see out/logistic-regression-performance.pdf).

We can see that as we increase the threshold, the precision and specificity increases, whereas the recall decreases.

This makes sense, because the closer our threshold is to 1, the more strict is our prediction criteria for classifing an example as 1. This means that there are going to be less false positives.

On the other hand, recall decreases because we are going to misclassify a lot of positive examples as negatives, thus making the number of false negatives higher.

2.6 Decision tree algorithm – training and test error rate, `cp` parameter

---

Firstly, I have created a decision tree with very low value of *cp* to get full *cptable*. Then, for each value of *cp* (there were four of them), I have created a decision tree model, plotted it and evaluated it on train and test sets. I have saved the accuracy values and analyzed them along with other parameters (*cp*, *nsplit*, *rel.error*, *xerror*, *xstd*):

CP	0.81410256	0.01923077	0.01282051	0.00000010
nsplit	0.00000000	1.00000000	4.00000000	5.00000000
rel.error	1.00000000	0.18589744	0.12820513	0.11538462
xerror	1.07692308	0.20512821	0.23717949	0.21794872
xstd	0.05665544	0.03436451	0.03662278	0.03529629
tree.accuracy.train	0.50318471	0.90764331	0.93630573	0.94267516
tree.accuracy.test	**0.48717949**	**0.88461538**	**0.85897436**	**0.87179487**
tree.error.train	0.49681529	0.09235669	0.06369427	0.05732484
tree.error.test	**0.51282051**	**0.11538462**	**0.14102564**	**0.12820513**

The value of *cp* parameter which produced the most accurate model was 0.01923077, with just one split – one decision depending on the value of *cylinders* attribute.

We can see that this is the best model based on test error, not based on training error. This is caused because of the data being overfitted when creating more splits.