Homework 4 - MATH 4322

Instructions

1. Due date: October 9, 2023
2. Scan or Type your answers and submit only one file. (If you submit several files only the recent one uploaded will be graded).
3. Preferably save your file as PDF before uploading. Submit in Canvas under Homework 4.
4. These questions are from *An Introduction to Statistical Learning with Applications in R* by James, et. al., chapter 5.

Problem 1

We will review k-fold cross-validation.

(a) Explain how k-fold cross-validation is implemented.

ANS:
K-fold Cross-Validation (CV) is an alternative to Leave-One-Out Cross-Validation (LOOCV) where the data is randomly divided into K subsets or folds of approximately equal size, denoted as n_k. The first fold serves as a validation set, while the method is trained on the remaining $k-1$ folds. Subsequently, the mean squared error (MSE1) is computed on the observations in the held-out fold. This entire process is repeated k times, with each iteration using a different group of observations as the validation set, resulting in k estimates of the test error, denoted as $MSE_1, MSE_2, \ldots, MSE_k$. The K-fold CV test error estimate is then calculated as the average of these k MSE values, represented as $\text{CV}(k) = \frac{1}{k}\sum_{j=1}^{k} MSE_j$. This method helps assess the model's performance and generalization ability while mitigating some of the issues associated with LOOCV, such as computational intensity.

(b) What are the advantages and disadvantages of k-fold cross-validation relative to:

i. The validation set approach?

Advantages:

The validation set approach is easy to implement.

It is conceptually simple.

Disadvantages:

A disadvantage is that Validation MSE can have high variability.

ii. LOOVC?

Advantages:

The model is fit only k « n times (where k is the number of folds and n is the number of data points).

The k-fold CV does not lose in estimation quality to LOOCV.

The variability in k-fold error estimates is trivial.

It is less computationally intensive than LOOCV for a smaller range of folds in CV.

This method works best for a large n (where n is the number of data points).

Disadvantages:

The disadvantage of the LOOCV is that it is computationally intensive, especially for a large dataset.

Problem 2

We will perform cross-validation on a simulated data set.

(a) Generate a simulated data set as follows:

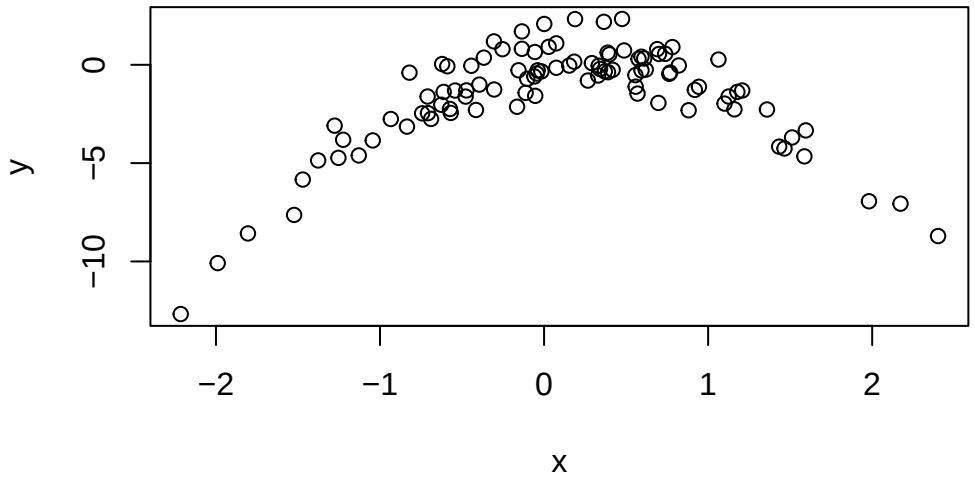
```
set.seed(1)
x=rnorm(100)
y=x-2*x^2+ rnorm(100)
```

In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

ANS: $n = 100, p = 2, y = x - 2x^2 + \epsilon$

(b) Create a scatterplot of X against Y . Comment on what you find.

```
plot(x, y)
```



This exhibits a non-linear, parabolic form.

(c) Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

i. $Y = \beta_0 + \beta_1 X + \epsilon$
ii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$
iii. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$
iv. $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$

Note: you might find it helpful to use the `data.frame()` function to create a single data set containing both X and Y.

```
library(boot)
Data = data.frame(x, y)
set.seed(10)

# Create an empty list to store the cross-validation results
cv.results = list()
```

```r
# Loop through polynomial degrees and perform cross-validation
for (degree in 1:4) {
  if (degree == 1) {
    model_formula = y ~ x
    model_label = "Linear"
  } else {
    model_formula = as.formula(paste("y ~ poly(x,", degree, ")"))
    model_label = paste("Polynomial (Degree", degree, ")")
  }

  glm.fit = glm(model_formula, data = Data)
  cv.error = cv.glm(Data, glm.fit)$delta

  # Store the cross-validation error along with the model label
  cv.results[[model_label]] = cv.error
}

# Display the cross-validation results
cv.results
```

```
$Linear
[1] 7.288162 7.284744

$`Polynomial (Degree 2 )`
[1] 0.9374236 0.9371789

$`Polynomial (Degree 3 )`
[1] 0.9566218 0.9562538

$`Polynomial (Degree 4 )`
[1] 0.9539049 0.9534453
```

(d) Repeat (c) using another random seed, and report your results. Are your results the same as what you got in (c)? Why?

```r
library(boot)
Data = data.frame(x, y)
set.seed(100)

# Create an empty list to store the cross-validation results
cv.results = list()
```

```r
# Loop through polynomial degrees and perform cross-validation
for (degree in 1:4) {
  if (degree == 1) {
    model_formula = y ~ x
    model_label = "Linear"
  }
  else {
    model_formula = as.formula(paste("y ~ poly(x,", degree, ")"))
    model_label = paste("Polynomial (Degree", degree, ")")
  }

  glm.fit = glm(model_formula, data = Data)
  cv.error = cv.glm(Data, glm.fit)$delta

  # Store the cross-validation error along with the model label
  cv.results[[model_label]] = cv.error
}

# Display the cross-validation results
cv.results
```

```
$Linear
[1] 7.288162 7.284744

$`Polynomial (Degree 2 )`
[1] 0.9374236 0.9371789

$`Polynomial (Degree 3 )`
[1] 0.9566218 0.9562538

$`Polynomial (Degree 4 )`
[1] 0.9539049 0.9534453
```

ANS: The outcomes remain identical because we applied LOOCV, which still evaluates each of the n folds, each containing a single observation.

(e) Which of the models in (c) had the smallest LOOCV error? Is this what you expected? Explain your answer.

ANS: The smallest LOOCV error is the quadratic polynomial, it corresponds to the authentic representation of Y in path B.

(f) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (c) using least squares. Do these results agree with the

conclusions drawn based on the cross-validation results?

```
summary(glm.fit)
```

```
Call:
glm(formula = model_formula, data = Data)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
poly(x, 4)1   6.18883    0.95905   6.453 4.59e-09 ***
poly(x, 4)2 -23.94830    0.95905 -24.971  < 2e-16 ***
poly(x, 4)3   0.26411    0.95905   0.275    0.784
poly(x, 4)4   1.25710    0.95905   1.311    0.193
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.9197797)

    Null deviance: 700.852  on 99  degrees of freedom
Residual deviance:  87.379  on 95  degrees of freedom
AIC: 282.3

Number of Fisher Scoring iterations: 2
```

The terms involving x and x^2 exhibit statistical significance, whereas the remaining terms do not. This aligns with the corss-validation results in part(c).

Problem 3

We will use a logistic regression to predict the probability of `default` using income and `balance` on the `Default` data set in the ISLR package. We will now estimate the test error of this logistic regression model using the validation set approach. Do not forget to set a random seed before beginning your analysis.

(a) Fit a logistic regression model that uses income and balance to predict default.

```
library(ISLR)
default.reg = glm(default ~ income + balance,
data = Default,family = "binomial")
summary(default.reg)
```

```
Call:
glm(formula = default ~ income + balance, family = "binomial",
    data = Default)

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.154e+01  4.348e-01 -26.545  < 2e-16 ***
income       2.081e-05  4.985e-06   4.174 2.99e-05 ***
balance      5.647e-03  2.274e-04  24.836  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2920.6  on 9999  degrees of freedom
Residual deviance: 1579.0  on 9997  degrees of freedom
AIC: 1585

Number of Fisher Scoring iterations: 8
```

(b) Using the validation set approach, estimate the test error of this model. In order to do this, you must perform the following steps:

i. Split the sample set into a training set and a validation set.

ii. Fit a multiple logistic regression model using only the training observations.

iii. Obtain a prediction of default status for each individual in the validation set by computing the posterior probability of default for that individual, and classifying the individual to the default category if the posterior probability is greater than 0.5.

iv. Compute the validation set error, which is the fraction of the observations in the validation set that are misclassified.

```r
#|echo: true
set.seed(2)
train = sample(1:nrow(Default),nrow(Default)/2)
default.test = Default[-train,]
default.train = Default[train,]
default.reg2 = glm(default ~ income + balance,
data = default.train,family = "binomial")
pred.test = predict(default.reg2,newdata = default.test,type = "response")
yhat = ifelse(pred.test<0.5,"No","Yes")
```

```
(conf.mat = table(yhat,default.test$default))
```

```
yhat    No  Yes
  No  4819  101
  Yes   18   62
```

```
# Assuming you have a confusion matrix named conf.mat
error_rate_string <- paste("Error rate: ", (conf.mat[1, 2] + conf.mat[2, 1]) / sum(co
error_rate_string
```

```
[1] "Error rate:  0.0238"
```

(c) Repeat the process in (b) three times, using three different splits of the observations into a training set and a validation set. Comment on the results obtained.

```
#|echo: true
# Set a seed for reproducibility
set.seed(2)

# Initialize an empty list to store the confusion matrices and error rates
results <- list()

# Perform the process three times with different splits
for (i in 1:3) {
  # Create three distinct random splits of the data
  set.seed(i)  # Use a different seed for each split
  train_indices <- sample(1:nrow(Default), nrow(Default) / 2)
  default.test <- Default[-train_indices,]
  default.train <- Default[train_indices,]

  # Fit a logistic regression model
  default.reg <- glm(default ~ income + balance,
                     data = default.train,
                     family = "binomial")

  # Generate predictions on the validation dataset
  pred.test <- predict(default.reg, newdata = default.test, type = "response")
  yhat <- ifelse(pred.test < 0.5, "No", "Yes")

  # Create a confusion matrix
```

```r
  conf.mat <- table(yhat, default.test$default)

  # Calculate the error rate and store it in the results list
  error_rate <- (conf.mat[1, 2] + conf.mat[2, 1]) / sum(conf.mat)
  error_rate_string <- paste("Error rate for Iteration", i, ":", error_rate)

  # Store the confusion matrix and error rate string in the results list
  results[[i]] <- list(confusion_matrix = conf.mat, error_rate = error_rate_string)

  # Print the confusion matrix and error rate for this iteration
  cat("Confusion Matrix for Iteration", i, ":\n")
  print(conf.mat)
  cat(error_rate_string, "\n\n")
}
```

```
Confusion Matrix for Iteration 1 :

yhat    No  Yes
  No  4824  108
  Yes   19   49
Error rate for Iteration 1 : 0.0254

Confusion Matrix for Iteration 2 :

yhat    No  Yes
  No  4819  101
  Yes   18   62
Error rate for Iteration 2 : 0.0238

Confusion Matrix for Iteration 3 :

yhat    No  Yes
  No  4822  109
  Yes   23   46
Error rate for Iteration 3 : 0.0264
```

#ANS: All of these remained within a narrow range, fluctuating between 2.5% and 3.0%.