Solutions 3

Jumping Rivers

Validation using bootstraping

1. Fit a linear regression model to the `cars2010` data set with `FE` as the response, using `EngDispl`, `NumCyl` and `NumGears` as predictors.[1]

[1] The data set can be loaded `data("FuelEconomy", package = "AppliedPredictiveModeling")`.

```
library(caret)
data(FuelEconomy, package = "AppliedPredictiveModeling")
m = train(FE ~ EngDispl + NumCyl + NumGears, data = cars2010,
    method = "lm")
```

2. What is the training error rate (RMSE) for this model?[2]

[2] Hint: The training error can be found by taking the square root of the average square residuals. The `sqrt()` and `resid()` functions may be useful.

```
sqrt(mean(resid(m)^2))
```

```
## [1] 4.589728
```

3. What is the estimated test error rate from bootstrap resampling?.

```
m$results["RMSE"]
```

```
##       RMSE
## 1 4.619096
```

4. How does this compare to the training error that we estimated above?

```
# most of the time the test error is higher,
# you can retrain a number of times to
# convince yourself
```

5. Experiment with adding terms to the model, transformations of the predictors and interactions say and use cross validation to estimate test error for each. What is the best model you can find?

Penalised regression

The `diabetes` data set in the `lars` package contains measurements of a number of predictors to model a response y, a measure of disease progression. There are other columns in the data set which contain interactions so we will extract just the predictors and the response. The data has already been normalized.

```
data(diabetes, package = "lars")
diabetesdata = cbind(y = diabetes$y, diabetes$x)
```

1. Try fitting a lasso, ridge and elastic net model using all of the main effects and pairwise interactions from each of the predictors.[3]

[3] Hint: $y \sim (.) \wedge 2$ is short hand for a model that includes main effect and pairwise interactions for each predictor.

```
m.lasso = train(y ~ (.)^2, data = diabetesdata,
    method = "lasso", tuneLength = 10)
m.ridge = train(y ~ (.)^2, data = diabetesdata,
    method = "ridge", tuneLength = 10)
m.enet = train(y ~ (.)^2, data = diabetesdata,
    method = "enet", tuneLength = 10)
```

1. Try to narrow in on the region of lowest RMSE for each model, don't forget about the `tuneGrid` argument to the train function.

2. We can view the coefficients via

```
coef = predict(m.lasso$finalModel,
mode = "fraction",
# which ever fraction was chosen as best
s = m.lasso$bestTune$fraction,
type = "coefficients"
)
```

3. How many features have been chosen by the `lasso` and `enet` models?

```
coef = predict(m.lasso$finalModel,
    mode = "fraction",
    s = m.lasso$bestTune$fraction,# which ever fraction was chosen as best
    type = "coefficients"
    )
sum(coef$coefficients != 0)
```

```
## [1] 36
```

```
coef = predict(m.enet$finalModel,
    mode = "fraction",
    s = m.enet$bestTune$fraction,# which ever fraction was chosen as best
    type = "coefficients"
    )
sum(coef$coefficients != 0)
```

```
## [1] 12
```

4. How do these models compare to a standard linear regression?

```
m = train(y ~ (.)^2, data = diabetesdata, method = "lm")
getTrainPerf(m)
```

```
##   TrainRMSE TrainRsquared TrainMAE method
## 1   61.2528     0.4045337 48.71708     lm

## better in terms of all performance metrics
```

5. Create a dotplot and parallel plot of the performance metrics?

```
res = resamples(list(lasso = m.lasso, ridge = m.ridge,
    enet = m.enet, lm = m))
dotplot(res)
parallelplot(res)
```