final_exam

Julene

2023-12-05

Climate Change Preliminaries

```
ols <- lm(y ~., data = co2)
summary(ols)$coefficients[,4]
```

```
## (Intercept)          x1          x2          x3          x4          x5
##  0.34277846  0.62164514  0.34048450  0.32234767  0.28131012  0.85685438
##          x6          x7
##  0.02323098  0.07822060
```

Looking at the coefficients of the OLS and their p-values, it seems possible that all the parameters other than x6 and x7 are 0, as their p-values are fairly high. (Assuming we are looking for significance level of 10%, if it is 5% I think x7 seems iffy too.)

Regularised Regressions

Since we want to reduce the number of variables in the model, we would want to use the lasso regression, as it will reduce some of the coefficients to 0.

Performing the regularisation with cross-validation on the entire set of data since I don't know what would be an appropriate test/train split,

```
cv.lasso <- cv.glmnet(X, Y,
                      alpha = 1,
                      nfolds=20,
                      standardize = TRUE)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
coef(cv.lasso)
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) 17.816883887
## x1           .
## x2           .
## x3           .
## x4           .
## x5           .
## x6           0.007214271
## x7           .
```

Well, I'm not sure what to do with this since basically all the coefficients are 0 except for x6 and we're supposed to have three variables so um. I'm just going to take the 3 that had the lowest p-values from the OLS.

```
ols2 <- lm(y ~x4 + x6 +x7, data = co2)
summary(ols2)$coefficients[,4]
```

```
##  (Intercept)           x4           x6           x7
## 3.371538e-01 4.671611e-01 9.812889e-05 3.737084e-02
```

Looking at the p-values, it seems x4 could still be 0. Let's remove it:

```
ols3 <- lm(y ~ x6 +x7, data = co2)
summary(ols3)$coefficients[,4]
```

```
##  (Intercept)           x6           x7
## 4.907239e-01 5.662641e-07 3.410006e-02
```

```
ols3$coefficients
```

```
## (Intercept)          x6          x7
##  2.52663030  0.01852197  2.18571951
```

```
summary(ols)
```

```
##
## Call:
## lm(formula = y ~ ., data = co2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.971  -4.681  -1.197   3.947  21.375
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 59.854132  61.514996   0.973   0.3428
## x1          -0.141682   0.282406  -0.502   0.6216
## x2          -0.252553   0.258300  -0.978   0.3405
## x3           0.864419   0.850724   1.016   0.3223
## x4          -0.419695   0.378477  -1.109   0.2813
## x5          -0.001778   0.009726  -0.183   0.8569
## x6           0.020523   0.008314   2.468   0.0232 *
## x7           2.035639   1.093560   1.861   0.0782 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.68 on 19 degrees of freedom
## Multiple R-squared:  0.7244, Adjusted R-squared:  0.6229
## F-statistic: 7.135 on 7 and 19 DF,  p-value: 0.0002993
```

Now, both x6 and x7 have p values lower than 0.05. Hence, I think this is the most appropriate model. The equation is y = 0.0185x6 + 2.19x7, rounding the coefficients to 3 s.f. The proportion of variance explained by this model is 72% (R^2 value when we look at the summary).

Colonisation and Diabetes

```
## 'data.frame':    768 obs. of  9 variables:
##  $ Pregnancies            : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                : int  148 85 183 89 137 116 78 115 197 125 ...
##  $ BloodPressure          : int  72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness          : int  35 29 0 23 35 0 32 0 45 0 ...
```

```
##  $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
```

Looking at the structure of the dataset, we see that all columns except "BMI" and "DiabetesPedigreeFunction" are "integers", while the aforementioned columns are "numeric".

```
tree_p <- rpart(Outcome ~.,
                    data=train,
                    method="class",   # "anova" for quantitative response
                    cp = 0, minsplit = 1)

tree_perf <- function(treemodel, test, class_column) {
  pred_class <- predict(treemodel, newdata=test, type='class')
  con <- confusionMatrix(pred_class, test[[class_column]])
  con$overall
}
tree_perf(tree_p, test, "Outcome")
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.6883117      0.3096750      0.6087863      0.7604240      0.6623377
## AccuracyPValue  McnemarPValue
##      0.2775305      0.8852339
```

The accuracy of the decision tree is about 69% (to 2 s.f.).

Ideally, I would use a for-loop to find and stire the accuracies of each forest, but I can't figure out what column or thing to call that would return me the accuracy. So, brute force it is.

```
set.seed(213)
T <- 500
for (m in 1:8) {
  return(randomForest(Outcome~., data = train, ntree = T, mtry = m))
}
forest <- randomForest(Outcome~., data = train, ntree = T, mtry = 1)
set.seed(213)
randomForest(Outcome~., data = train, ntree = T, mtry = 1) #25.24
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 1)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 25.24%
## Confusion matrix:
##     0   1 class.error
## 0 347  51   0.1281407
## 1 104 112   0.4814815
```

```
set.seed(213)
randomForest(Outcome~., data = train, ntree = T, mtry = 2) #26.38
```

```
##
## Call:
```

```
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 2)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 26.38%
## Confusion matrix:
##     0   1 class.error
## 0 331  67   0.1683417
## 1  95 121   0.4398148
```

```r
set.seed(213)
randomForest(Outcome~., data = train, ntree = T, mtry = 3) #24.59
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 3)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 24.59%
## Confusion matrix:
##     0   1 class.error
## 0 336  62   0.1557789
## 1  89 127   0.4120370
```

```r
set.seed(213)
randomForest(Outcome~., data = train, ntree = T, mtry = 4) #25.73
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 4)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 25.73%
## Confusion matrix:
##     0   1 class.error
## 0 329  69   0.1733668
## 1  89 127   0.4120370
```

```r
set.seed(213)
randomForest(Outcome~., data = train, ntree = T, mtry = 5) #25.41
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 5)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 25.41%
## Confusion matrix:
##     0   1 class.error
```

```
## 0 333  65   0.1633166
## 1  91 125   0.4212963
```

```
set.seed(213)
randomForest(Outcome~., data = train, ntree = T, mtry = 6) #26.22
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 6)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 26.22%
## Confusion matrix:
##     0   1 class.error
## 0 327  71   0.1783920
## 1  90 126   0.4166667
```

```
set.seed(213)
randomForest(Outcome~., data = train, ntree = T, mtry = 7) #24.92
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 7)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 24.92%
## Confusion matrix:
##     0   1 class.error
## 0 333  65   0.1633166
## 1  88 128   0.4074074
```

```
set.seed(213)
randomForest(Outcome~., data = train, ntree = T, mtry = 8) #25.57
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 8)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 25.57%
## Confusion matrix:
##     0   1 class.error
## 0 330  68   0.1708543
## 1  89 127   0.4120370
```

So, it seems that using only 3 variables considered at each split would be optimal, as it has the the lowest error rate and thus the highest accuracy rate.

Growing this forest,

```r
set.seed(213)
forest <- randomForest(Outcome~., data = train, ntree = T, mtry = 3)
print(forest)
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = train, ntree = T,      mtry = 3)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 24.59%
## Confusion matrix:
##     0   1 class.error
## 0 336  62   0.1557789
## 1  89 127   0.4120370
```

We see that it has an accuracy rate of 75.41%.

OK so this doesn't work:

```r
fit.glm_skin <- glm(Outcome ~., data = train,
               family = binomial())
fit.glm_skin
```

```
##
## Call:  glm(formula = Outcome ~ ., family = binomial(), data = train)
##
## Coefficients:
##               (Intercept)               Pregnancies                   Glucose
##                -8.2130001                 0.1183929                 0.0352638
##             BloodPressure             SkinThickness                   Insulin
##                -0.0130875                -0.0009585                -0.0009091
##                       BMI  DiabetesPedigreeFunction                       Age
##                 0.0860673                 0.7810790                 0.0152288
##
## Degrees of Freedom: 613 Total (i.e. Null);  605 Residual
## Null Deviance:       796.4
## Residual Deviance: 583.5     AIC: 601.5
```

```r
train_no_skin <- train[, -4]
fit.glm_no_skin <- glm(Outcome ~., data = train_no_skin,
               family = binomial())

range(diabetes$Pregnancies)
```

```
## [1]  0 17
```

```r
range(diabetes$Glucose)
```

```
## [1]   0 199
```

```r
range(diabetes$BloodPressure)
```

```
## [1]   0 122
```

```r
range(diabetes$SkinThickness)
```

```
## [1]  0 99
```

```
range(diabetes$Insulin)
```

```
## [1]   0 846
```

```
range(diabetes$BMI)
```

```
## [1]  0.0 67.1
```

```
range(diabetes$DiabetesPedigreeFunction)
```

```
## [1] 0.078 2.420
```

```
range(diabetes$Age)
```

```
## [1] 21 81
```

```
pregnancies_grid <- seq(0, 17, length=20)
glucose_grid <- seq(0, 199, length = 20)
bloodPressure_grid <- seq(0, 122, length=20)
skinThickness_grid <- seq(0, 99, length = 20)
insulin_grid <- seq(0, 846, length = 20)
bmi_grid <- seq(0, 67.1, length = 20)
function_grid <- seq(0.078, 2.420, length = 20)
age_grid <- seq(21, 81, length = 20)

#setting up the grid
#multi_variable_grid_skin <- expand.grid(Pregnancies = pregnancies_grid, Glucose = glucose_grid, BloodP
#multi_variable_grid <- expand.grid(Pregnancies = pregnancies_grid, Glucose = glucose_grid, BloodPressu
#predicted_probabilities_skin <- predict(fit.glm_skin, newdata = multi_variable_grid_skin, type = "resp
#predicted_probabilities_no_skin <- predict(fit.glm_no_skin, newdata = multi_variable_grid, type = "res
```

Just in case my code doesn't work, here's a sketch of what we're supposed to do for this question:

1. Conduct hypothesis test by shuffling the predicted responses for the models with and without SkinThickness. The test-statistic should be difference in RMSE. Calculate the difference between the two RMSEs and plot a histogram. If the original difference is very far from the middle of the histogram, then it is significant and we cannot remove SkinThickness. But otherwise, we can.

```
fit.glm <- glm(Outcome ~ Pregnancies + Glucose + BloodPressure + Insulin + BMI + DiabetesPedigreeFunctio
               family = binomial())
summary(fit.glm)$coefficients[,4]
```

```
##              (Intercept)              Pregnancies                  Glucose
##             5.158094e-26             9.515978e-04             1.931996e-17
##            BloodPressure                  Insulin                      BMI
##             1.959080e-02             2.847660e-01             9.265106e-08
## DiabetesPedigreeFunction                      Age
##             1.507614e-02             1.339849e-01
```

Looking at the p-values, we can probably remove insulin first. Age is probably the next to go. Let's repeat and check until all p-values <0.05.

```
fit.glm <- glm(Outcome ~ Pregnancies + Glucose + BloodPressure + BMI + DiabetesPedigreeFunction + Age, d
               family = binomial())
summary(fit.glm)$coefficients[,4]
```

```
##              (Intercept)              Pregnancies                  Glucose
##             3.277832e-26             8.515978e-04             3.825455e-18
```

```
##             BloodPressure                       BMI DiabetesPedigreeFunction
##              1.663826e-02              1.249841e-07              2.022116e-02
##                       Age
##              1.100891e-01
```

```
fit.glm <- glm(Outcome ~ Pregnancies + Glucose + BloodPressure + BMI + DiabetesPedigreeFunction, data =
               family = binomial())
summary(fit.glm)$coefficients[,4]
```

```
##              (Intercept)              Pregnancies                  Glucose
##              1.918882e-26              2.048618e-06              1.645084e-19
##             BloodPressure                       BMI DiabetesPedigreeFunction
##              3.043235e-02              2.543016e-07              1.550446e-02
```

```
fit.glm$coefficients
```

```
##              (Intercept)              Pregnancies                  Glucose
##              -7.77290156               0.14790614               0.03494044
##             BloodPressure                       BMI DiabetesPedigreeFunction
##              -0.01199183               0.08104640               0.76495809
```

Ok this new one looks good. The equation of the resultant model is probability = sigma(0.148 pregnancies + 0.0349 glucose - 0.0120 bloodPressure + 0.0810 BMI + 0.765 DiabetesPedigreeFunction) = 1/(1+e^-the equation we just wrote).

```
odds <- 1/(1+exp(-0.148*3 - 0.0349*120 + 0.0120*70 - 0.0810*32 - 0.765*0.35))
```

The odds are 99.9% for this person.

```
predicted_probabilities <- predict(fit.glm, newdata = test, type = "response")
```

Ok I ran out of time so to outline the next steps for d and e pls give me some method marks T_T:

d. Convert the probabilities to classes using the ptoclass function from the last lab. Find the best threshold to use by finding the accuracies for each threshold and pick the one with the highest acccuracy. Then, do the barplot to compare the accuracies of each model.

e. The most performant would be the one with the highest accuracy. This method may or may not outperform the other two methods for different classification problems, as each method has its own pros and cons. Each method has different ways of determining the classes that suit diff problems.