Will it Rain Tomorrow?

Christopher Heather

15/11/2021

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(warning = FALSE)
knitr::opts_chunk$set(message = FALSE)
```

1. Introduction

Australia is the world's driest inhabited contintent. Watching the weather and hoping for rain is therefore somewhat of a national passtime. The Australian Bureau of Meteorology (BOM) is the national weather service, conducting meteorological observations at weather stations across the country. These data are made available publicly online. The "Rain in Australia" dataset has been compiled from this source and made available by Joe Young on Kaggle (https://www.kaggle.com/jsphyg/weather-dataset-rattle-package).

This project aims to use the rainfall dataset to predict whether it will rain tomorrow in a specific location using machine learning (and without the benefit of complex meteorological models). Not an easy task, given the geographical and climactic diversity of the Australian continent, ranging from dry desert to temperate and alpine environments to monsoonal tropics.

In order to achieve this goal the following steps were followed: - Data import and cleaning - Data exploration - Pre-processing - Development of predictive models, including logistic regression and random forests. - Evaluation of predictive accuracy of these models.

As the outcome (Rain Tomorrow) is a binary categorical ("Yes" or "No"), overall Accuracy was used to evaluate the performance of the models.

2. Data import and cleaning

2.1 Importing the data

The data set "Rain in Australia" was downloaded from Kaggle and made available on a Git repository for this project. After reviewing data types, coercion of character columns into factors was performed.

```
### Data import

# Load required packages

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(RANN)) install.packages("RANN", repos = "http://cran.us.r-project.org")
```

```
library(tidyverse)
library(lubridate)
library(caret)
library(randomForest)
library(RANN)


# Australian rainfall data https://www.kaggle.com/jsphyg/weather-dataset-rattle-package
# (accessed 15/11/21)

# rainAUS.csv uploaded to Git repository
# https://github.com/csheather/harvardX_data_science_capstone_ausRain

# dowload data file from Github repository

dl <- tempfile()
download.file("https://raw.githubusercontent.com/csheather/harvardX_data_science_capstone_ausRain/main/

# Read file into data frame
ausrain <-
  read_csv(dl)

# Coerce character columns into factors
ausrain <- ausrain %>%
  mutate(RainToday = as.factor(RainToday),
         RainTomorrow = as.factor(RainTomorrow),
         Location = as.factor(Location),
         WindGustDir = as.factor(WindGustDir),
         WindDir9am = as.factor(WindDir9am),
         WindDir3pm = as.factor(WindDir3pm))

# Clean up
rm(dl)
```

The resulting data frame contains 145460 observations of 23 variables of the following types.

```
## spec_tbl_df [145,460 x 23] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Date         : Date[1:145460], format: "2008-12-01" "2008-12-02" ...
##  $ Location     : Factor w/ 49 levels "Adelaide","Albany",..: 3 3 3 3 3 ...
##  $ MinTemp      : num [1:145460] 13.4 7.4 12.9 9.2 17.5 ...
##  $ MaxTemp      : num [1:145460] 22.9 25.1 25.7 28 32.3 ...
##  $ Rainfall     : num [1:145460] 0.6 0 0 0 1 ...
##  $ Evaporation  : num [1:145460] NA NA NA NA NA ...
##  $ Sunshine     : num [1:145460] NA NA NA NA NA ...
##  $ WindGustDir  : Factor w/ 16 levels "E","ENE","ESE",..: 14 15 16 5 14 ...
##  $ WindGustSpeed: num [1:145460] 44 44 46 24 41 ...
##  $ WindDir9am   : Factor w/ 16 levels "E","ENE","ESE",..: 14 7 14 10 2 ...
##  $ WindDir3pm   : Factor w/ 16 levels "E","ENE","ESE",..: 15 16 16 1 8 ...
##  $ WindSpeed9am : num [1:145460] 20 4 19 11 7 ...
##  $ WindSpeed3pm : num [1:145460] 24 22 26 9 20 ...
##  $ Humidity9am  : num [1:145460] 71 44 38 45 82 ...
##  $ Humidity3pm  : num [1:145460] 22 25 30 16 33 ...
##  $ Pressure9am  : num [1:145460] 1008 1011 ...
```

```
##  $ Pressure3pm  : num [1:145460] 1007 1008 ...
##  $ Cloud9am     : num [1:145460] 8 NA NA NA 7 ...
##  $ Cloud3pm     : num [1:145460] NA NA 2 NA 8 ...
##  $ Temp9am      : num [1:145460] 16.9 17.2 21 18.1 17.8 ...
##  $ Temp3pm      : num [1:145460] 21.8 24.3 23.2 26.5 29.7 ...
##  $ RainToday    : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 ...
##  $ RainTomorrow : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 ...
```

2.2 Creating Month and Year columns

Column 1 of the resulting table contains the date stamp for each observation. This is unlikely to enlightening, however month and year may hold some predictive value once the model is constructed. The `Date` column was therefore converted into separate `Month` and `Year` columns.

```
### Create Month and Year Columns

ausrain <- ausrain %>%
  mutate(Month = as.factor(month(Date)),
         Year = as.factor(year(Date))) %>%
  select(-Date)
```

2.3 Removing NA from outcome variable

The column `RainTomorrow` was identified as holding the outcome variable to be predicted. This is a two level factor with the values `No` and `Yes`. Unfortunately this column contains 3267 missing values. The rows containing missing values in this column were removed as they are not useful in constructing or validating our model.

```
### Identify and remove NA in ausrain$RainTomorrow

ausrain <- filter(ausrain, !is.na(RainTomorrow))
```

2.4 Split data into working and validation sets.

After creating the new columns and removing missing values from out outcome variable the dataset now contains 142193 observations of 24 variables.

The data set was now divided to create a withholding subset of data, `validation`, which will be used to test the final performance of the models.

```
### Creation of validation set

# validation set is 10% of initial ausrain set

set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = ausrain$RainTomorrow, times = 1, p = 0.1, list = FALSE)
ausrain <- ausrain[-test_index,]
validation <- ausrain[test_index,]
rm(test_index)
```

The working data (`ausrain`) now contains 127973 observations, while the `validation` set contains 14220 observations.

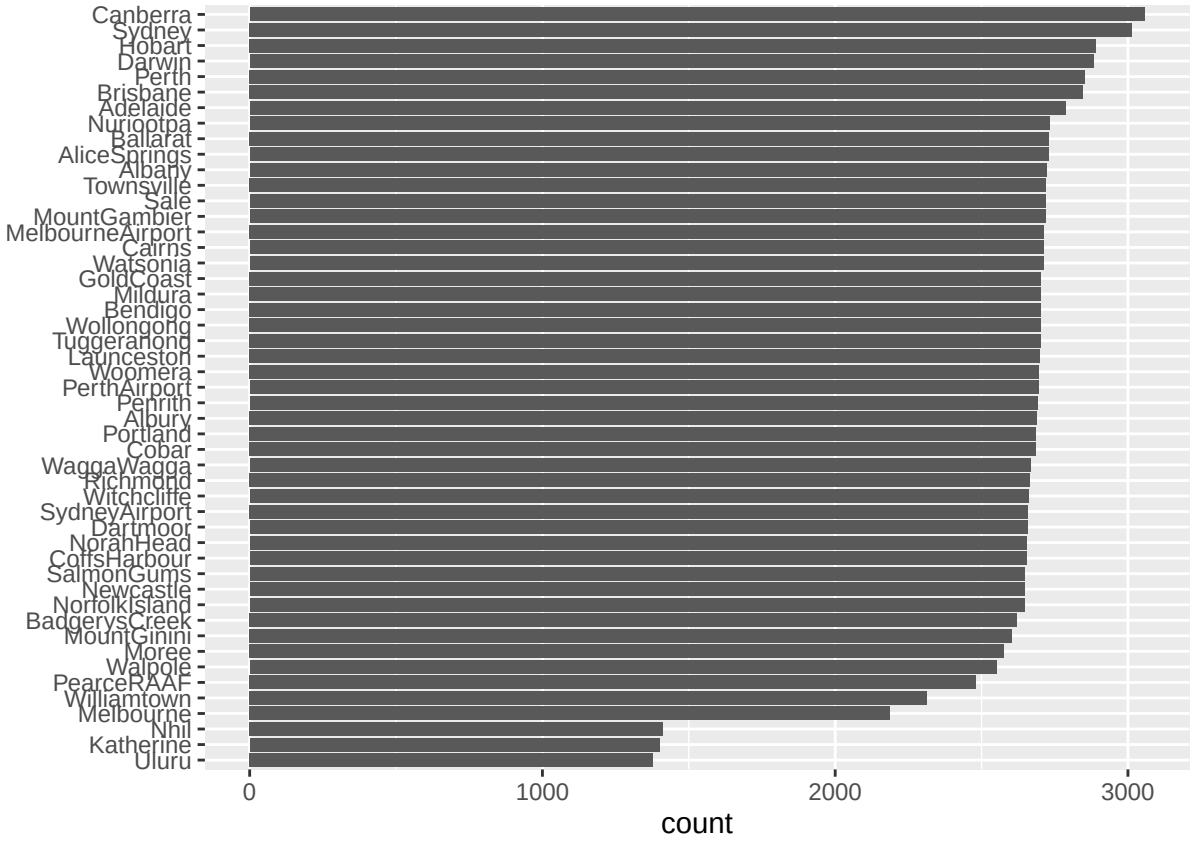
3. Data Exploration and Parameter selection

3.1 Location and time data

The `ausrain` data set contains observation from 49 locations in Australia over a period of 11 years.

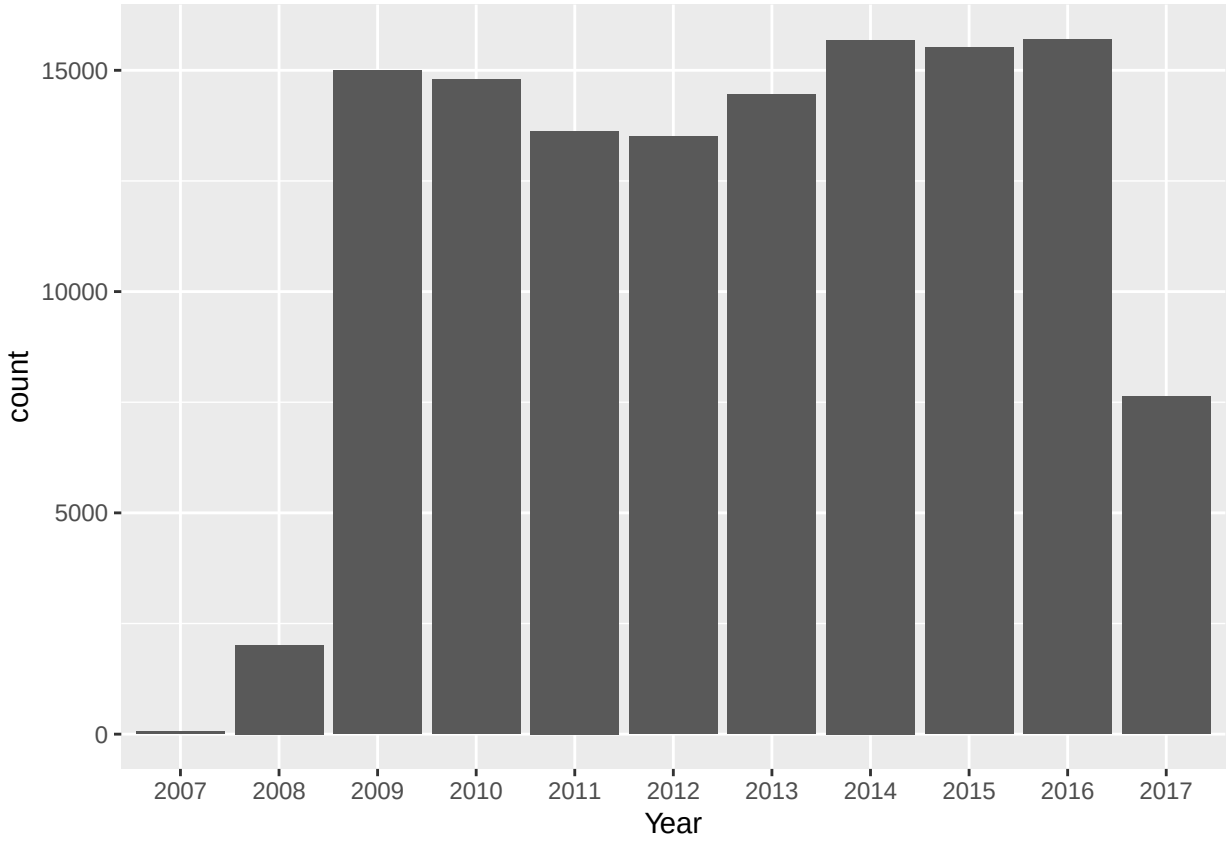
From the graphs below we can see that observations are fairly evenly distributed, with the exception of several central Australian locations ("Katherine", "Nhil", and "Uluru"), which have approximately half the number of observations, though there appear to be no extreme outliers.

```
### Distribution of Location observations

ausrain %>% group_by(Location) %>% summarise(count = n()) %>%
  ggplot(aes(reorder(Location, count), count)) +
  geom_col() +
  coord_flip()+
  xlab("")
```



However when looking at observations by year, it is apparent that the years 2007 and 2008 are very much under-represented.

```
### Distribution of Year observations

ausrain %>% group_by(Year) %>% summarise(count = n()) %>%
  ggplot(aes(Year, count)) +
  geom_col()
```



3.2 Significance tests for predictors

Since the data contains a significant number of potential parameters, it would be desirable to drop any predictors for which there is no significant difference in samples grouped by the outcome variable `RainTomorrow`. As out data contains both factors and numeric data, and the outcome is a binomial, the chi-squared test was used on categorical variables and the T-test on numeric variables. A significance level of $p<0.05$ was used as the threshold to reject the null hypothesis (no difference between groups).

```
### Significance tests

# Find column names by type

factor_names <- ausrain %>% select(where(is.factor)) %>% colnames()
numeric_names <- ausrain %>% select(where(is.numeric)) %>% colnames()


# Perform chi-squared test on categorical parameters

p_values_chisq <- sapply(factor_names, function(x){
  test <- ausrain %>% select(RainTomorrow, x) %>%
    table() %>%
    chisq.test()
  return(test$p.value)
})
```

```r
# Number of non-significant parameters - categorical

sum(p_values_chisq >= 0.05)
```

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

```r
# perform T-test on continuous parameters

p_values_T <- sapply(ausrain[,numeric_names], function(x) {
  test <- t.test(x ~ ausrain$RainTomorrow, var.equal = TRUE)
  return(test$p.value)
})

#Number of non-significant parameters - numeric

sum(p_values_T >= 0.05)
```

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

```r
#Clean up

rm(factor_names, p_values_chisq, p_values_T)
```

Performing the chi-squared test and T-test on our data suggests that all our potential parameters show some relation to our outcome variable `RainTomorrow`.

3.3 Variables with large amount of missing data

Next potential parameters were assessed for missing data. Large amounts of missing data may bias models. In addition, some algorithms, such as tree based methods, do not tolerate any missing data. Variables with more than 30% of missing values were dropped from the data set.

```r
### Find large number of NA

too_many_NA <- ausrain %>% summarise(across(.fns = ~mean(is.na(.)) >= 0.3))

colnames(ausrain[,which(too_many_NA == TRUE)])
```

```
## [1] "Evaporation" "Sunshine"    "Cloud9am"    "Cloud3pm"
```

Removal of these columns may impact the accuracy of the final models, but on review of the variable names, none stood out as likely to be critical to incorporate in the model. They were therefore removed.

```r
### Remove large number of NA

ausrain <- ausrain %>% select(!which(too_many_NA == TRUE))

rm(too_many_NA)
```

3.4 Near zero variance

Next variables were checked for near zero variance (NZV), where one value predominates within a particular parameter to the extent that predictive power is lost. The default passed to the `nearZeroVar` function is 95/5 for the ratio of the most common value over the next most common.

```
### Near Zero Variance

nearZeroVar(ausrain)
```

```
## integer(0)
```

This indicates that there appear to be no columns with NZV in the data set.

3.5 Highly correlated variables

The numeric data were next assessed for highly correlated variables, some of which could be dropped as parameters for the final model.

```
### Highly corrlated variables

# Names of numeric variables (needs to be updated because some removed)

numeric_names <- ausrain %>% select(where(is.numeric)) %>% colnames()

#Correlation matrix

correlations <- cor(ausrain[,numeric_names], use = "na.or.complete")

#Find highly correlated variables

highlyCorrelated <- findCorrelation(correlations, cutoff=0.9)
colnames(correlations)[highlyCorrelated]
```

```
## [1] "Temp9am"     "MaxTemp"     "Pressure3pm"
```

The `highlyCorrelated` function checks a correlation matrix and returns a vector with suggested columns to be removed. The suggested columns were removed from the data set.

```
### Remove highly correlated variables

ausrain <- select(ausrain, -colnames(correlations)[highlyCorrelated])

rm(highlyCorrelated, correlations)
```

3.6 Linear dependencies

Linear dependencies may exist in the data, where two variables show high linear correlation. This is particularly problematic for linear regression models, however for any model, some of these variables can be removed without significant impact on predictions.

```
### Find linear dependencies

# update numeric_names

numeric_names <- ausrain %>% select(where(is.numeric)) %>% colnames()

# Find linear dependencies

linearCombos <- findLinearCombos(na.omit(ausrain[,numeric_names]))

linearCombos
```

```
## $linearCombos
## list()
##
## $remove
## NULL
```

```
rm(numeric_names, linearCombos)
```

This shows that there are no linear dependencies in the data set.

After this initial feature selection the resulting `ausrain` dataset contained 127973 observations over 17 columns.

```
### Data structure

str(ausrain, vec.len = 2)
```

```
## tibble [127,973 x 17] (S3: tbl_df/tbl/data.frame)
##  $ Location     : Factor w/ 49 levels "Adelaide","Albany",..: 3 3 3 3 3 ...
##  $ MinTemp      : num [1:127973] 13.4 7.4 12.9 9.2 17.5 ...
##  $ Rainfall     : num [1:127973] 0.6 0 0 0 1 ...
##  $ WindGustDir  : Factor w/ 16 levels "E","ENE","ESE",..: 14 15 16 5 14 ...
##  $ WindGustSpeed: num [1:127973] 44 44 46 24 41 ...
##  $ WindDir9am   : Factor w/ 16 levels "E","ENE","ESE",..: 14 7 14 10 2 ...
##  $ WindDir3pm   : Factor w/ 16 levels "E","ENE","ESE",..: 15 16 16 1 8 ...
##  $ WindSpeed9am : num [1:127973] 20 4 19 11 7 ...
##  $ WindSpeed3pm : num [1:127973] 24 22 26 9 20 ...
##  $ Humidity9am  : num [1:127973] 71 44 38 45 82 ...
##  $ Humidity3pm  : num [1:127973] 22 25 30 16 33 ...
##  $ Pressure9am  : num [1:127973] 1008 1011 ...
##  $ Temp3pm      : num [1:127973] 21.8 24.3 23.2 26.5 29.7 ...
##  $ RainToday    : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 ...
##  $ RainTomorrow : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 ...
##  $ Month        : Factor w/ 12 levels "1","2","3","4",..: 12 12 12 12 12 ...
##  $ Year         : Factor w/ 11 levels "2007","2008",..: 2 2 2 2 2 ...
```

4. Preprocessing

Next, further data manipulation was performed. It should be noted that the `caret` package allows preprocessing to be performed within the `train()` function, however for the purposes of clarity in this report

all pre-processing steps were performed prior to running the `train` function. Additionally, all pre-processing, such as data imputation was performed on the `ausrain` data set. Subsequently the values derived from this process were applied to impute missing values in the `validation` set. No data from the `validation` set were used in this process.

4.1 Data Imputation

The data sets still contained a significant number of missing values. Across the data 55863 values were missing.

Data for numeric parameters was imputed using the mean for each parameter using the `caret::preProcess()` function.

```
### Imputing numeric values

# median imputation of numeric values

numimpute_preProcess_object <- preProcess(as.data.frame(ausrain),
                                  method = "medianImpute")

# impute numeric values in ausrain

ausrain <- predict(numimpute_preProcess_object, newdata = as.data.frame(ausrain))

# use same object to impute numeric values in validation

validation <- predict(numimpute_preProcess_object, newdata = as.data.frame(validation))

# Clean up

rm(numimpute_preProcess_object)
```

The data now contained 22137 missing values in the parameters which are factors and were ignored by the `medianImpute` method.

There are several options for imputing factor values. The simplest is mode imputation, however this may significantly bias the data set. KNN imputation is more reliable, however is computationally expensive on large data sets.

The proportion of rows with missing values after imputing numerical values was relatively at 0.13. It was therefore decided that, rather than pursuing KNN imputation or potentially problematic mode imputation, the remaining observations with missing values would be dropped during training.

5. Model construction

A number of machine learning models were constructed in order to tackle this problem, and their accuracy was assessed.

5.1 Random Guess

The first model was a random guess, based on proportion of rainy days.

```
### Random Guess

#overall likelihood of rain the next day

p_tomorrow <- mean(ausrain$RainTomorrow == "Yes")

#Guess randomly using p_tomorrow

set.seed(1999, sample.kind = "Rounding")
p_hats <- sample(x = c("Yes", "No"),
                 size = nrow(ausrain),
                 replace = TRUE,
                 prob = c(p_tomorrow, 1-p_tomorrow))

# assess accuracy

guess_accuracy <-confusionMatrix(as.factor(p_hats), ausrain$RainTomorrow)$overall[["Accuracy"]]

# Start a table with results

results_table <- tribble(~"Model", ~"Accuracy",
                         "Random Guess", guess_accuracy)

# Clean up

rm(p_tomorrow, p_hats)
```

This model had an accuracy of 0.6507388.

5.2 RainToday

Another approach would be to look at whether it is raining today and deduce that it will probably rain tomorrow as well.

```
### Guess on RainToday

# select RainToday and RainTomorrow, remove NA

df <- ausrain %>% select(RainToday, RainTomorrow) %>% na.omit()

# Assess accuracy

guess_accuracy <- confusionMatrix(df$RainTomorrow, df$RainToday)$overall[["Accuracy"]]

# Add to table

results_table <- add_row(results_table,
                         Model = "RainToday",
                         Accuracy = guess_accuracy)

# Clean up

rm(df)
```

The accuracy of this single predictor was already better at 0.7616274.

5.2 Logistic regression

The modest improvement with one predictor is encouraging, however a model incorporating all out remaining predictors might yield even better accuracy. As out data contains a mixture of numerical and categorical parameters, and out outcome variable is binomial, the analysis may lend itself to a logistic regression model. This was trained using the `caret` package. The `glm` function does not have any hyperparameters to tune, so tuning was not incorporated into the model.

```
### Generalised Linear Model

# Train logistic regression model. NA are excluded from training data set.

ausrain_glm <- train(RainTomorrow ~., data = ausrain,
                     method = "glm",
                     na.action = na.omit,
                     family = "binomial")

# Generate predictions from validation data using trained model

predictions_glm <- predict(ausrain_glm,
                           newdata = validation,
                           type = "raw",
                           na.action = na.pass)

# Determine accuracy of predictions

glm_accuracy <- confusionMatrix(predictions_glm, validation$RainTomorrow)$overall[["Accuracy"]]

# Add result to the table

results_table <- add_row(results_table,
                         Model = "Logistic Regression",
                         Accuracy = glm_accuracy)

rm(guess_accuracy, predictions_glm, ausrain_glm)
```

Fitting the logistic regression model substantially improved the accuracy of the predictions to 0.8493555.

5.3 Trees

Tree based models may also represent a useful tool to further enhance predictive accuracy in this situation, as the response variable is binomial.

5.3.1 Regression Tree

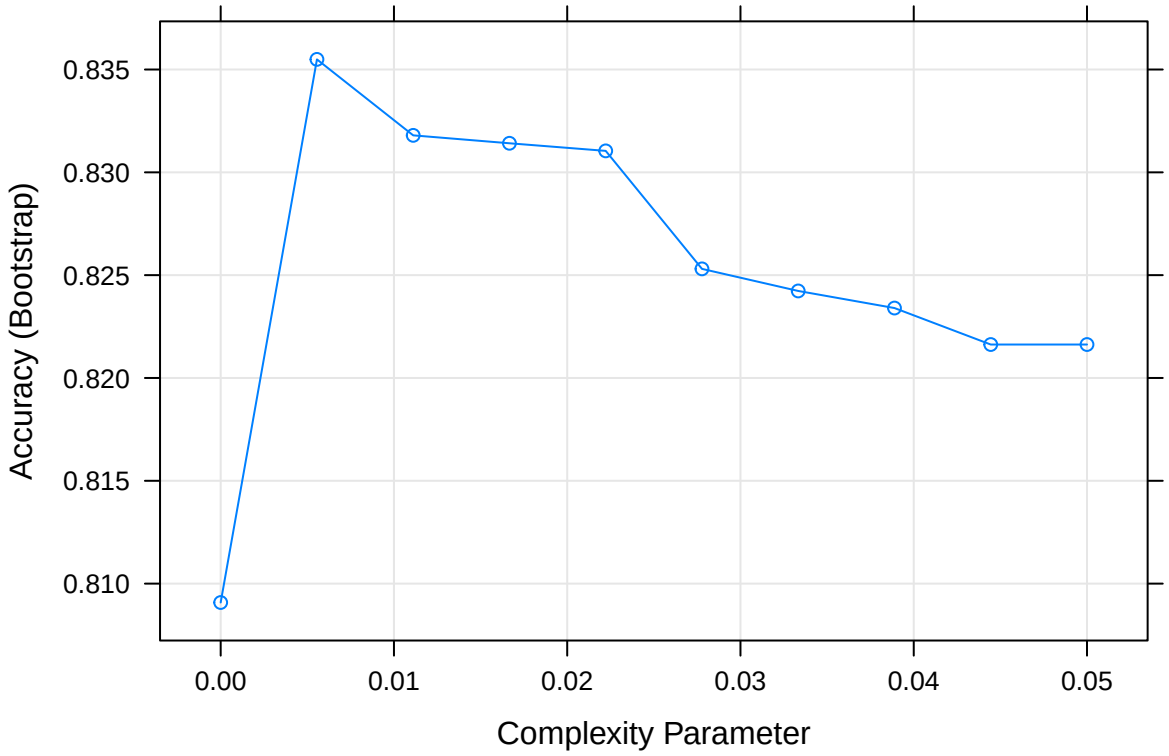
A regression tree was trained using `rpart`.

The tree was tuned using the complexity parameter `cp` over 10 values between 0 and 0.05. All other hyperparameters were accepted as per the default in `rpart.control`.

```
### Regression Tree

# Train tree

train_rpart <- train(RainTomorrow~., data = ausrain,
      method = "rpart",
      tuneGrid = data.frame(cp = seq(0, 0.05, len = 10)),
      na.action = na.omit)

# Use training set to predict outcomes in validation set

predict_rpart <- predict(train_rpart,
                         newdata = validation,
                         na.action = na.pass)

# assess accuracy

rpart_accuracy <- confusionMatrix(predict_rpart,validation$RainTomorrow)$overall[["Accuracy"]]

# Add result to the table
results_table <- add_row(results_table,
                         Model = "Regression Tree",
                         Accuracy = rpart_accuracy)
```

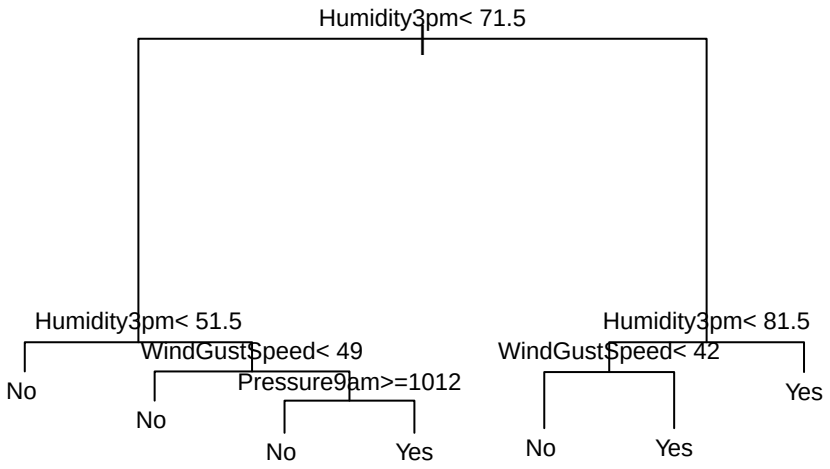
Examining the `train_rpart` object shows the complexity parameter 0.0055556 resulted in the greatest accuracy.

```
### Regression tree hyperparameter

plot(train_rpart)
```



The final decision tree included only 4 predictors.

```
### Plotting final tree

plot(train_rpart$finalModel, margin = 0.1)
text(train_rpart$finalModel, cex = 0.75)
```




```
rm(train_rpart, glm_accuracy)
```

Somewhat disappointingly, the accuracy of the Regression Tree approach was 0.8407405, which was slightly worse than logistic regression.

5.3.2 Random Forest

Extending this approach, a random forest model was trained, averaging multiple decision trees in order to improve predictive accuracy. The model was set to use 5 fold cross validation and the number of randomly sampled variables `mtry` was set to 5, which approximately corresponds to the square root of the number of predictors. Hyperparameters could have been tuned within the `train()` function, however this was not done on this occasion due to computational cost.

In the prediction step, missing values in validation were imputed using `na.roughfix()`, which replaces missing data in factors with the mode, while numeric data in out data sets had already been imputed.

```
### Random Forest

# Set number of cross validations

control <- trainControl(method="cv", number = 5)
grid <- data.frame(mtry = 5) # ~sqrt(p)

# Train using ausrain set
```

```
train_rf <-  train(RainTomorrow ~., data = ausrain,
                   method = "rf",
                   ntree = 100,
                   trControl = control,
                   tuneGrid = grid,
                   na.action = na.omit)

# Predict using validation set

predict_rf <- predict(train_rf,
                      newdata = validation,
                      na.action = na.roughfix)

# Detemine Accuracy

rf_accuracy <- confusionMatrix(predict_rf, validation$RainTomorrow)$overall[["Accuracy"]]

# Add result to the table

results_table <- add_row(results_table,
                         Model = "Random Forest",
                         Accuracy = rf_accuracy)

#Clean up

rm(control, train_rf, predict_rf, rpart_accuracy)
```

Even without tuning, the random forest model significantly improved accuracy to 0.9217371, greater than 90% for the first time.

6. Results

For this project, several machine learning models were trained in order to predict next-day rainfall from the "Rain in Australia" data set. The Accuracy metric for the models are detailed in the table below.

```
### Results table

knitr::kable(results_table)
```

Model	Accuracy
Random Guess	0.6507388
RainToday	0.7616274
Logistic Regression	0.8493555
Regression Tree	0.8407405
Random Forest	0.9217371

This shows that logistic regression and regression trees were able to improve substantially on simple guesses, while a random forest approach was able to increase predictive accuracy to over 90%.

7. Conclusions, Limitations and Future work

While this result is respectable, further improvements might have been made with some additional adjustments to the methodology.

Firstly, additional data imputation using KNN imputation may have increased both the observations available and the quality of the imputed data. This was not performed on this occasion due to computational costs.

Second, additional parameter selection/ reduction, such as principal component analysis, might have optimised computation times. The `pca` function is available within the `caret::preProcess` function, but was not applied on this occasion. Similarly, converting factors to dummy variables for regression tree analysis was not performed, but may have improved data quality for these approaches.

Third, the regression tree and random forest models contain several hyperparameters, which could have been tuned during model training in order to find more optimal settings. Again, in the interest of manageable computation times on home computers, extensive tuning was not performed on this occasion.

Predicting rain in Australia is not always straightforward. In this project machine learning principles were applied to a large meteorological data set with good results, reaching a predictive accuracy of 0.9217371.