DS 740: Final Project

Eli Bolotin

7/26/2019

Contents

1 Load packages and functions

```r
if (!require("pacman")) install.packages("pacman")
pacman::p_load(randomForest, mltools, data.table, e1071, caret, doParallel, R.utils,
↪  neuralnet, QuantPsych, prcomp, pROC)
```

1.1 Create helper functions

Create helper functions to perform preprocessing tasks:

1. Check skewness of predictors
2. Print confusion matrix and prediction accuracy
3. Generate cv groups

```r
check_skewness <- function(df, numeric_cols) {
  skew_list <- c()
  par(mfrow=c(5,4), mar=c(3,1,1,1))
  for(i in numeric_cols){
    feature_skewness <- skewness(na.omit(df[, i]))
    skew_list <- c(skew_list, feature_skewness)
    title_and_skewness <- paste(i,': ', round(feature_skewness,2))
    hist(df[,i], main = title_and_skewness)
  }
  skewness_df <- data.frame("skewness" = skew_list, "features" = numeric_cols)
  return(skewness_df)
}

get_accuracy <- function(predictions, actuals, print_cm = "yes") {
  cm <- addmargins(table(predictions, actuals))
  if(print_cm == "yes") {
    print(cm)
  }
  diag.cm <- diag(cm)
  len_diag <- length(diag.cm)
  accuracy <- sum(diag.cm[-len_diag])/diag.cm[len_diag]
  return(as.numeric(accuracy))
}

generate_cvgroups <- function(k, n) {
  if ((n %% k) == 0) {
      groups = rep(1:k, floor(n/k))
  } else {
      groups = c(rep(1:k, floor(n/k)), (1:(n %% k)))
  }
  cvgroups = sample(groups, n)
  return(cvgroups)
}

test_empty_col <- function(x) {
  empty_col = 0
  if(all(x %in% 0)) {
    empty_col = empty_col + 1
  }
  return(empty_col)
}
```

2 Data loading and evaluation

```
data <- read.csv("weatherAUS.csv", na.strings=c("", "NA"))
dim(data)
```

```
## [1] 142193     24
```

2.1 Examine data

```
head(data)
```

```
##         Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-01   Albury    13.4    22.9      0.6          NA       NA
## 2 2008-12-02   Albury     7.4    25.1      0.0          NA       NA
## 3 2008-12-03   Albury    12.9    25.7      0.0          NA       NA
## 4 2008-12-04   Albury     9.2    28.0      0.0          NA       NA
## 5 2008-12-05   Albury    17.5    32.3      1.0          NA       NA
## 6 2008-12-06   Albury    14.6    29.7      0.2          NA       NA
##   WindGustDir WindGustSpeed WindDir9am WindDir3pm WindSpeed9am
## 1           W            44          W        WNW           20
## 2         WNW            44        NNW        WSW            4
## 3         WSW            46          W        WSW           19
## 4          NE            24         SE          E           11
## 5           W            41        ENE         NW            7
## 6         WNW            56          W          W           19
##   WindSpeed3pm Humidity9am Humidity3pm Pressure9am Pressure3pm Cloud9am
## 1           24          71          22      1007.7      1007.1        8
## 2           22          44          25      1010.6      1007.8       NA
## 3           26          38          30      1007.6      1008.7       NA
## 4            9          45          16      1017.6      1012.8       NA
## 5           20          82          33      1010.8      1006.0        7
## 6           24          55          23      1009.2      1005.4       NA
##   Cloud3pm Temp9am Temp3pm RainToday RISK_MM RainTomorrow
## 1       NA    16.9    21.8        No     0.0           No
## 2       NA    17.2    24.3        No     0.0           No
## 3        2    21.0    23.2        No     0.0           No
## 4       NA    18.1    26.5        No     1.0           No
## 5        8    17.8    29.7        No     0.2           No
## 6       NA    20.6    28.9        No     0.0           No
```

2.2 Examine column types

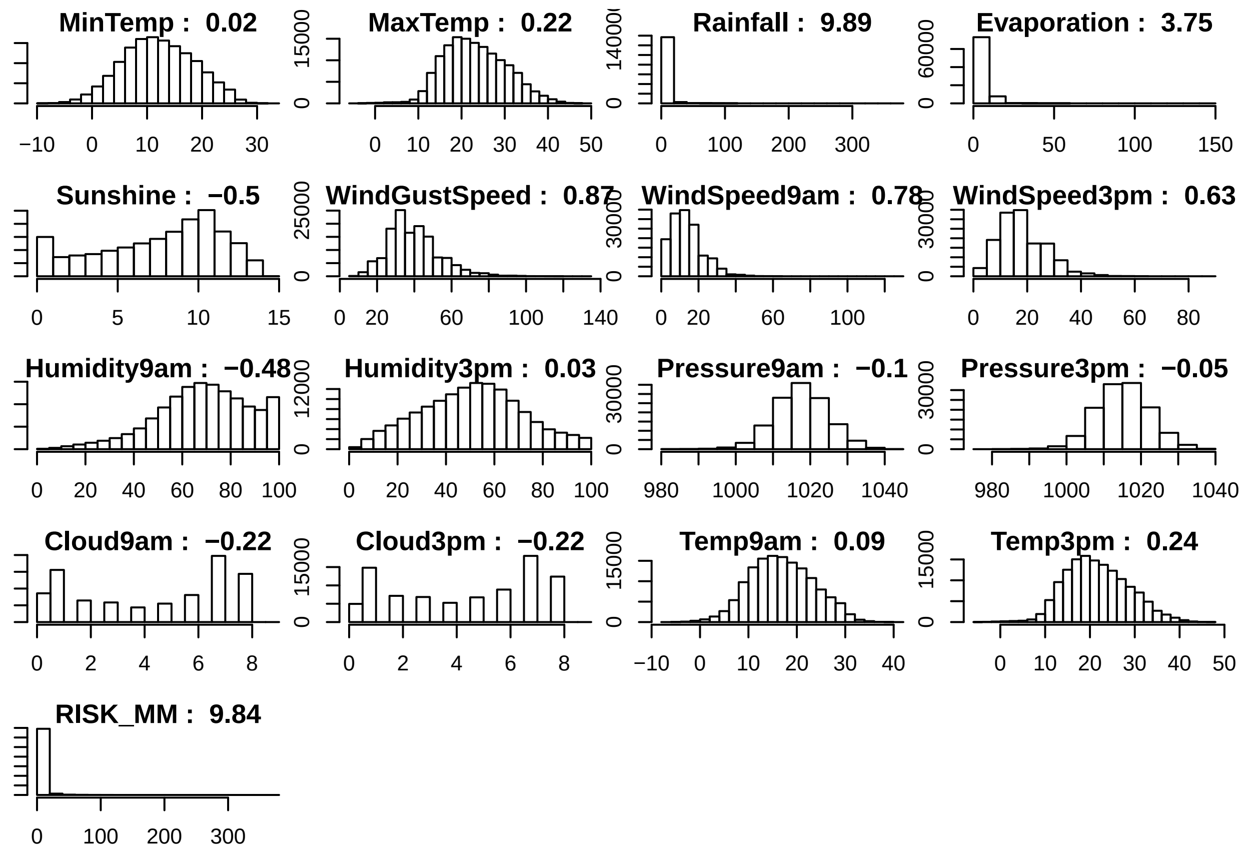
```
## Factor columns:  7
```

```
## [1] "Date"          "Location"      "WindGustDir"   "WindDir9am"
## [5] "WindDir3pm"    "RainToday"     "RainTomorrow"
## Numeric columns:  17
```

```
##  [1] "MinTemp"       "MaxTemp"       "Rainfall"      "Evaporation"
##  [5] "Sunshine"      "WindGustSpeed" "WindSpeed9am"  "WindSpeed3pm"
##  [9] "Humidity9am"   "Humidity3pm"   "Pressure9am"   "Pressure3pm"
## [13] "Cloud9am"      "Cloud3pm"      "Temp9am"       "Temp3pm"
## [17] "RISK_MM"
```

2.3 Examine skewness of numeric variables

```
skewness_of_data <- check_skewness(data, numeric.cols)
```



Data appears mostly Gaussian.

3 Data preprocessing and transformation

3.1 Create predictor to indicate percent of prior 10 days rain

I do this first (before dropping NAs) because the data is chronologically ordered by day.

```
n_rows <- nrow(data)
prior_days_rain <- rep(NA, n_rows)
days_prior = 10
```

```r
for(i in 1:n_rows) {
  if(i == 1) {
    percent_rain <- 0
    next
  } else if (i - 1 < days_prior) {
    window_start = 1
    window_stop = i - 1
  } else {
    window_start = i - days_prior
    window_stop = i - 1
  }
  tbl <- table(data$RainToday[window_start:window_stop])
  percent_rain <- round(tbl["Yes"]/sum(tbl), 2)
  prior_days_rain[i] <- percent_rain
}

prior_days_rain <- replace_na(prior_days_rain)
data["prior_days_rain"] <- prior_days_rain
```

3.2 Create new predictors to capture time lag of *RainToday*

```r
n_rows <- nrow(data)
lag_1_rain <- rep(NA, n_rows)
lag_2_rain <- rep(NA, n_rows)
lag_3_rain <- rep(NA, n_rows)

for(i in 4:n_rows) {
  lag_1 <- data$RainToday[i - 1]
  lag_2 <- data$RainToday[i - 2]
  lag_3 <- data$RainToday[i - 3]
  lag_1_rain[i] <- lag_1
  lag_2_rain[i] <- lag_2
  lag_3_rain[i] <- lag_3
}

lag_1_rain <- replace_na(lag_1_rain)
lag_2_rain <- replace_na(lag_2_rain)
lag_3_rain <- replace_na(lag_3_rain)

data["lag_1_rain"] <- as.factor(ifelse(lag_1_rain==2, "Yes", "No"))
data["lag_2_rain"] <- as.factor(ifelse(lag_2_rain==2, "Yes", "No"))
data["lag_3_rain"] <- as.factor(ifelse(lag_3_rain==2, "Yes", "No"))

data <- data[-c(1:3),]
```

3.3 Drop Risk_MM (correlated to response) and date (captured via new predictors)

```r
'%ni%' <- Negate('%in%')
df_names <- colnames(data)
filtered_names <- df_names[df_names %ni% c("RISK_MM","Date")]
data <- data[, filtered_names]
```

3.4 Evaluate variables with high missingness (>= 30%)

```r
n_rows <- dim(data)[1]
n_cols <- dim(data)[2]
missingness <- list()
i = 1

for(col in 1:n_cols) {
  num_na <- length(which(is.na(data[,col])))
  if(num_na/n_rows >= .30) {
    missingness[i] <- col
    i = i + 1
  }
}

missingness <- as.numeric(missingness)
missingness_cols <- colnames(data)[missingness]
cat("Variables with 30% or more missingness:\n", missingness_cols)
```

```
## Variables with 30% or more missingness:
##  Evaporation Sunshine Cloud9am Cloud3pm
```

Despite the high level of missingness, these are potentially important predictors. It's worth keeping them and removing a higher percentage of total NAs (because of these columns) - if we have an abundance of data afterwards. To find out, let's check NA count.

3.5 Check NA count

```r
total_na_rows <- length(which(apply(data, 1, function(x) (any(x %in% c("", "n/a") |
↪  is.na(x))))))
cat("Total NA rows:", total_na_rows)
cat("\nTotal rows after NA removal: ", n_rows - total_na_rows)
```

Total NA rows: 85770 Total rows after NA removal: 56420

There is an abundance of observations left for analysis after removing all rows with NAs (many caused by columns with high missingness).

3.6 Omit NAs from data

```r
# omit nas
data <- droplevels(na.omit(data))
n_rows <- dim(data)[1]
```

3.7 Review column types and number of levels per factor

```r
col.types <- sapply(data, class)

factor.cols <- names(col.types[col.types=="factor"])
numeric.cols <- names(col.types[col.types!="factor"])

factor.levels <- list()
i <- 0

for(col in factor.cols) {
  factor.levels[col] <- length(levels(data[,col]))
  i = i + 1
}

# Get total levels
sum(as.numeric(factor.levels))
```

```
## [1] 84
```

There are 84 levels of categorical predictors... One-hot coding categorical predictors will result in an additional ~84 extra features in our data.

3.8 One-hot-code factors and create feature sets by type

```r
# get all factors and exclude the response, raintomorrow
non_target <- which(factor.cols != "RainTomorrow")

x.factors <- data[, factor.cols[non_target]]

# create dataframe of one-hot-coded factors
x.factors.coded <- one_hot(as.data.table(data[, c(factor.cols[non_target])]))

# create dataframe of numeric predictors
x.numeric <- data[, numeric.cols]

# create target vector
y <- as.factor(data$RainTomorrow)
```

3.9 Create preprocessing, train, and test groups

The reason for a preprocessing group is to have data that is separate from training data that can be used for feature selection without introducing additional bias to the model (to avoid overfitting).

```r
set.seed(3)

n_rows <- nrow(data)

# Sample percent of data for training and testing
traintest_subset <- round(n_rows * 0.95)
sub_sample <- sample(1:n_rows, traintest_subset)
```

```r
# create train/test sampled data
x.numeric.traintest <- x.numeric[sub_sample,]
x.factors.coded.traintest <- x.factors.coded[sub_sample,]
y.traintest <- y[sub_sample]

# create sampled data for preprocessing (feature selection)
x.numeric.preprocessing <- x.numeric[-sub_sample,]
x.factors.coded.preprocessing <- x.factors.coded[-sub_sample,]
y.preprocessing <- y[-sub_sample]
```

3.10 Scale data

```r
set.seed(3)

# sample 80% of data for training
total_traintest <- nrow(x.numeric.traintest)
training_rows <- round(total_traintest * 0.80)
train_indices = sample(1:total_traintest, training_rows)

# scale train/test data
x.numeric.train.std <- scale(x.numeric.traintest[train_indices,])
x.numeric.valid.std = scale(x.numeric.traintest[-train_indices,], center =
↪  attr(x.numeric.train.std, "scaled:center"), scale = attr(x.numeric.train.std,
↪  "scaled:scale"))
```

3.11 Finalize *preprocessed*, *train*, and *test* datasets

```r
set.seed(3)

# create train/test sets
train_data <- data.frame(raintomorrow = y.traintest[train_indices], x.numeric.train.std,
↪  x.factors.coded.traintest[train_indices,])
test_data <- data.frame(raintomorrow = y.traintest[-train_indices], x.numeric.valid.std,
↪  x.factors.coded.traintest[-train_indices,])

# preprocessing data for feature selection
preprocessed_data <- data.frame(raintomorrow = y.preprocessing,
↪  scale(x.numeric.preprocessing), x.factors.coded.preprocessing)

# check number of training features
n_train_features <- ncol(train_data)
```

4 Feature selection

After hot-coding, the training set has 100 features (up from 23 originally). Using all of these features is not likely necessary to achieve near maximum prediction accuracy, and worse yet, is very computationally inefficient. To figure out which features to keep, we perform feature selection.

4.1 Identify highly correlated variables

```
# create (Y,X) correlation matrix
corr.matrix <- cor(preprocessed_data[,-1])

# find highly correlated vars to remove
correlated_predictors <- findCorrelation(corr.matrix, cutoff=0.5)

# check out names
colnames.pp.data <- colnames(preprocessed_data[,-1])
most_correlated_features <- colnames.pp.data[correlated_predictors]

cat("Total of", length(most_correlated_features), "correlated features:\n")
```

```
## Total of 15 correlated features:
```

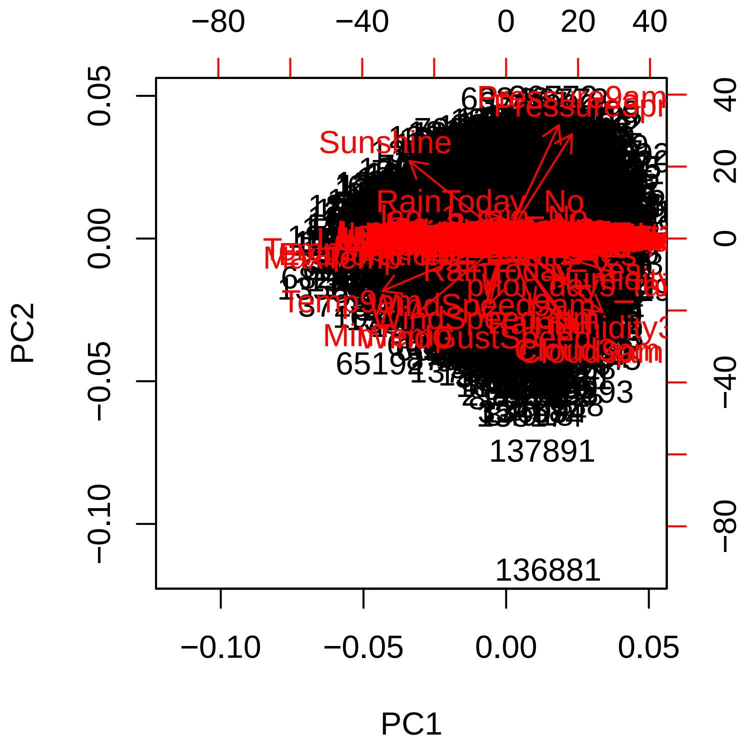
```
most_correlated_features
```

```
##  [1] "Temp3pm"       "MaxTemp"       "Temp9am"       "Humidity3pm"
##  [5] "Humidity9am"   "Sunshine"      "RainToday_No"  "RainToday_Yes"
##  [9] "Pressure3pm"   "Cloud3pm"      "lag_1_rain_No" "WindGustSpeed"
## [13] "WindSpeed3pm"  "lag_2_rain_No" "lag_3_rain_No"
```

Multicollinearity indicates that certain variables are redundant. Due to the high correlation of these predictors, perhaps they could be "combined", so to speak, to be represented in a lower dimensional space that contains a significant amount of variation in the data. This is the idea behind Principal Component Analysis, which we perform next.

4.2 Perform PCA

```
pca_analysis <- prcomp(preprocessed_data[,-1], scale = FALSE)

# biplot of PCA
biplot(pca_analysis)
```



```r
# proportion of variance explained by first 10 principal components
vjs = pca_analysis$sdev^2
pve = vjs/sum(vjs)
sum(pve[1:10])
```

```
## [1] 0.7524348
```

The first 10 principal components explain 75% of total variability.

In the plot above, the red arrows represent the first two PC loading vectors [1]. PC2 (2nd principal component, y-axis) puts most of its emphasis on features that have vertical direction (such as Pressure9am, Pressure3am), while PC1 puts most if its emphasis on features with horizontal direction (such as Temp9a, Max Temp, Evaporation). The black points represent PC scores. The location of score on the plot relative to the loading vectors indicates the magnitude of weather characteristics (features) for a specific day in the dataset.

4.3 Perform feature selection with RFE

Below, we perform backwards selection with recursive feature elimination (based on predictor importance ranking). Random forest is used to perform 5-fold CV on 8 possible predictor subsets. Note that preprocessing is performed on the dataset with PCA.

```r
start_time <- Sys.time()

# define cores and register parallel computation operation
cl <- makePSOCKcluster(detectCores())
registerDoParallel(cl)
set.seed(6)
```

```r
# tuning hyperparameters
p <- ncol(preprocessed_data)-1

# define mtry subsets
subset_sizes <- seq(6,p,2)[1:8]

# set seed for reproducible results
seeds <- vector(mode = "list", length = 6)
for(i in 1:5) seeds[[i]]<- rep(3,9)
seeds[[6]]<- 3

# perform RFE using best mtry from rf_model
rfe_control <- rfeControl(functions=rfFuncs, method="cv", number=5, seeds = seeds,
↪  returnResamp = T, saveDetails = T)
rfe_results <- rfe(raintomorrow~., data=preprocessed_data, sizes = subset_sizes,
↪  preProcess = "pca",  rfeControl=rfe_control)

stopCluster(cl)

end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 56.77447 secs
```

4.4 Plot RFE results and review best features

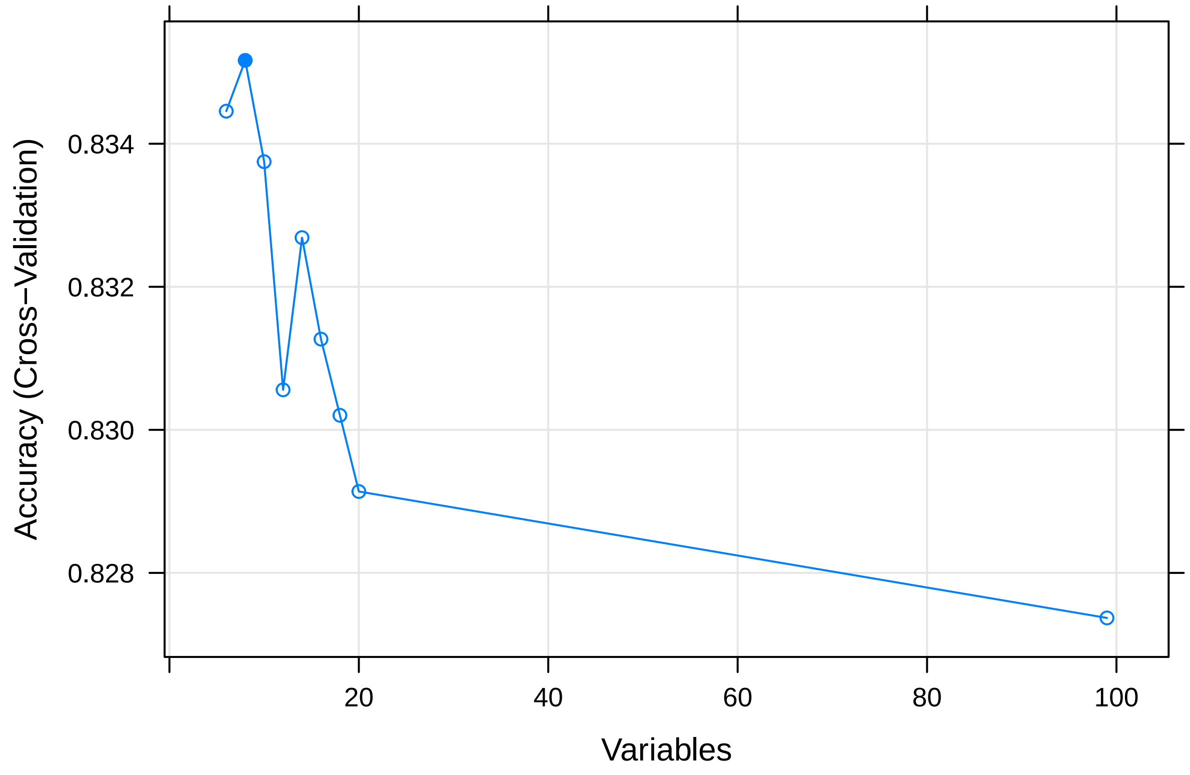
```r
par(mfrow=c(1,3))

print(rfe_results)
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (5 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          6   0.8345 0.4780   0.005275 0.02438
##          8   0.8352 0.4812   0.006500 0.01763        *
##         10   0.8337 0.4804   0.010244 0.02901
##         12   0.8306 0.4688   0.006841 0.02141
##         14   0.8327 0.4760   0.008828 0.03117
##         16   0.8313 0.4689   0.007987 0.03204
##         18   0.8302 0.4653   0.003333 0.02067
##         20   0.8291 0.4591   0.003906 0.01917
##         99   0.8274 0.4400   0.007860 0.03820
##
## The top 5 variables (out of 8):
##    Humidity3pm, Sunshine, WindGustSpeed, Pressure3pm, Cloud3pm
```

```r
predictors(rfe_results)
```

```
## [1] "Humidity3pm"   "Sunshine"      "WindGustSpeed" "Pressure3pm"
## [5] "Cloud3pm"      "Pressure9am"   "Rainfall"      "Humidity9am"
```

```r
plot(rfe_results, type=c("g", "o"))
```



```r
cat("Optimal count of variables is: ", length(rfe_results$optVariables))
```

```
## Optimal count of variables is:  8
```

4.5 Important notes

- By using PCA for feature selection, we are sacrificing a small amount of model accuracy to improve model interpretability. This is to say: had we not used PCA preprocessing, the "optimal" model would have many more predictor variables while only improving prediction accuracy by a very small amount.
- The time lags of RainToday (i.e. lag_1_rain) and prior 10 days rain (prior_days_rain) do not increase predictive accuracy. Therefore, predicting *raintomorrow* is not a matter of cyclical or time-based autoregressive forecasting.

5 Model training

Recursive feature eliminaion indicates that for this dataset, prediction accuracy does not increase with the increase of predictor subset sizes. Rather, the most parsimonious model with 6 predictors (acc = 0.8345) is actually slightly better than the most complex model of 99 predictors (acc = 0.8274). We will use the optimal number of predictors (of 8 predictors) for training. They are:

- Humidity3pm, Sunshine, WindGustSpeed, Pressure3pm, Cloud3pm, Pressure9am, Rainfall, Humidity9am

5.1 Define model formula

```
predictors = rfe_results$optVariables[1:rfe_results$bestSubset]
new_formula <- paste("raintomorrow~",paste(predictors, collapse="+"), sep="")
new_formula <- as.formula(new_formula)
```

5.2 Take smaller sample of train/test data

- Taking a smaller sample of train/test data for computational efficiency. The full sets will be used in final model fitting.

```
set.seed(3)

n_rows_traindata <- nrow(train_data)
n_rows_testdata <- nrow(test_data)
sample_size_train <- round(n_rows_traindata * 0.15)
sample_size_test <- round(n_rows_testdata * 0.15)
small_sample_train <- sample(1:n_rows_traindata, sample_size_train)
small_sample_test <- sample(1:n_rows_testdata, sample_size_test)
train_data_small <- train_data[small_sample_train,]
test_data_small <- test_data[small_sample_test,]
```

Size of full training data: 42879 records

Size of full test data: 10720 records

Size of small sample of training data: 6432 records

Size of small sample of test data: 1608 records

5.3 Define function to perform CV-10

Training performed with Bagging and linear SVM

```
cv.rfsvm <- function(k, input_formula, training_data, mtry) {
  num_train <- nrow(training_data)
  # store predictions on training data
  bagging.single.cv.predictions = rep(NA, num_train)
  svm.single.cv.predictions = rep(NA, num_train)

  # create CV groups
  cvgroups = generate_cvgroups(k, num_train)

  # 10-fold CV
  iteration_num = 1

  for(i in 1:k) {
    printf("\rIteration %d out of %d", iteration_num, k, file = "")

    # define train/val indices/subset
    train_indices = (cvgroups != i)
    val_indices = (cvgroups == i)

    bagging.fit <- randomForest(input_formula, data = training_data[train_indices,], mtry
↪   = mtry)
    svm.fit = svm(input_formula, data = training_data[train_indices,], kernel = "linear",
↪   scale=FALSE, type="C-classification")

    # perform predictions
    x_val <- training_data[val_indices,c(predictors)]
    bagging.single.cv.predictions[val_indices] = predict(bagging.fit, x_val,
↪   type="class")
    svm.single.cv.predictions[val_indices] = predict(svm.fit, x_val, type="class")
    iteration_num = iteration_num + 1
  }
  output = list(bagging.single.cv.predictions, svm.single.cv.predictions)
  return(output)
}
```

5.4 Perform CV-10 on compact training data

```
set.seed(3)

start_time <- Sys.time()

single.cv.10 = cv.rfsvm(10, new_formula, train_data_small, mtry = length(predictors))
bagging.predictions = single.cv.10[[1]]
svm.predictions = single.cv.10[[2]]

printf("\n")
end_time <- Sys.time()

end_time - start_time
```

Calculation time: 37.4 minutes

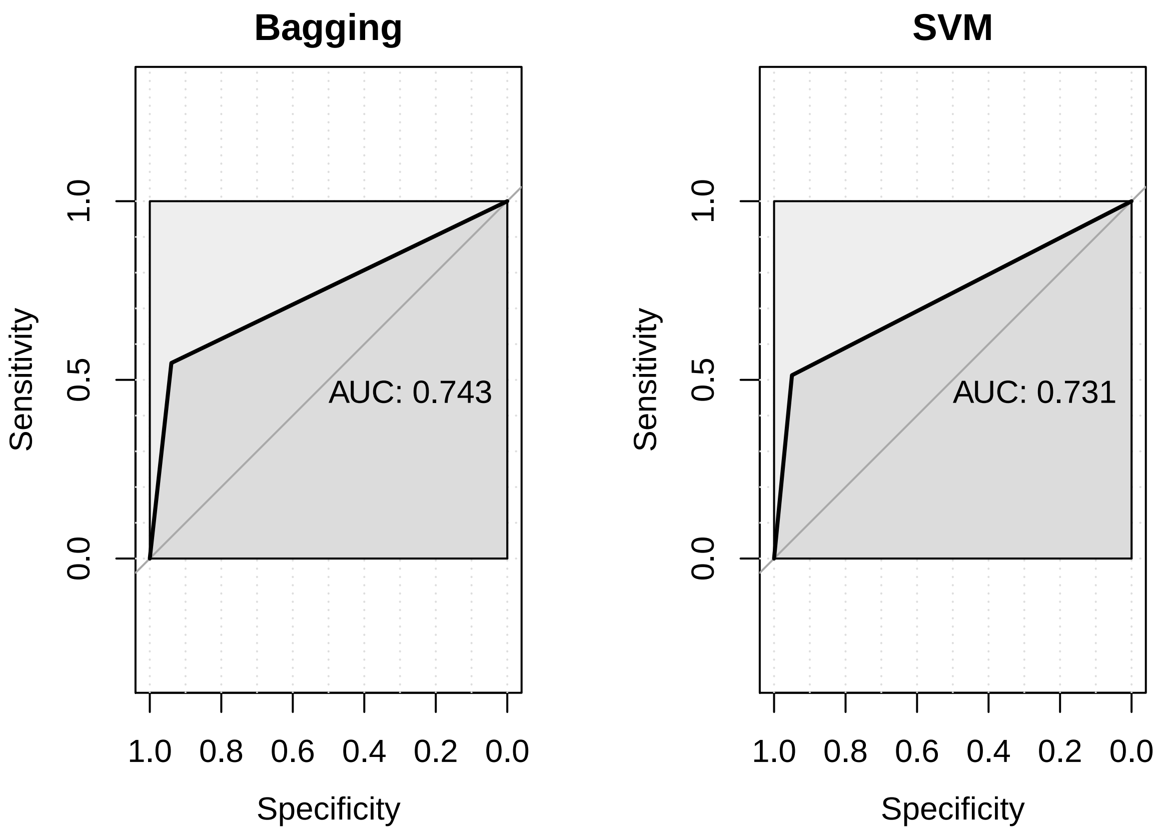
5.4.1 CV-10 score on compact training data

```
## CV-10 accuracy using top 8 predictors:

## Bagging:

##              actuals
## predictions    No  Yes  Sum
##         1    4730  633 5363
##         2     304  765 1069
##         Sum  5034 1398 6432

##
```

```
## Accuracy of bagging:  0.8543221

##
##
## SVM:

##            actuals
## predictions   No  Yes  Sum
##         1   4782  681 5463
##         2    252  717  969
##         Sum 5034 1398 6432

##
## Accuracy of SVM:  0.854944
```

Both models produce comparable CV-10 scores.

5.4.2 Plot ROC curves for models



A perfect model (100% accuracy) would achieve an area under the curve (AUC) of 1. A more realistic goal is to achieve an AUC of >= 0.90%. To do so with the models above, both models need to do a better job at classifying true positives (true positive rate = sensitivity). Bagging results in a higher AUC.

5.5 Perform Double CV-10 (nested 10-fold cross validation)

Training is performed with bagging and linear SVM. For every fold of the outer loop, the inner loop performs 10-fold cross validation. The best model of the inner CV-10 is then trained on the outer folds and used to make predictions.

```r
start_time <- Sys.time()
set.seed(3)

# set hyperparameters
k = 10
p = ncol(train_data_small)-1
n <- nrow(train_data_small)
n_models = 2

# create sampling groups
cvgroups.out = generate_cvgroups(k, n)

# define containers for predictions
double.cv.predictions = rep(NA, n)

# perform outer cross-validation loop
iteration_num = 1

for (i in 1:k)  {
  train_indices_cv = (cvgroups.out != i)
  validation_indices_cv = (cvgroups.out == i)
  validation_data_cv = train_data_small[validation_indices_cv,]

  # re-sample training observations
  resampled_cvgroups = generate_cvgroups(k, n)

  # define container for results
  cv10.model.comparison <- matrix(NA, nr = k, nc = n_models)

  for(j in 1:k) {
    # print countdown
    printf("\rIteration %d out of %d", iteration_num, k^2, file = "")

    # define validation data
    train_indices_resampled = (resampled_cvgroups != j)
    validation_indices_resampled = (resampled_cvgroups == j)
    validation_data_resampled = train_data_small[validation_indices_resampled,]
    validation_data_resampled_y = train_data_small[validation_indices_resampled,
↪   "raintomorrow"]

    # fit models
    rf.fit = randomForest(new_formula, data = train_data_small, subset =
↪   train_indices_resampled)
    svm.fit = svm(new_formula, data = train_data_small, kernel="linear", scale=FALSE,
↪   type="C-classification", subset = train_indices_resampled)

    # make predictions
    rf.predictions = predict(rf.fit, validation_data_resampled, type="class")
    svm.predictions = predict(svm.fit, validation_data_resampled, type="class")

    cv10.model.comparison[j, 1] = get_accuracy(rf.predictions,
↪   validation_data_resampled_y, "no")
```

```
    cv10.model.comparison[j, 2] = get_accuracy(svm.predictions,
↪   validation_data_resampled_y, "no")

    iteration_num = iteration_num + 1
  }

  if(i == 1) {
    cv10.model.total.comparison <- cv10.model.comparison
  } else {
    cv10.model.total.comparison <- rbind(cv10.model.total.comparison,
↪   cv10.model.comparison)
  }

  highest_accuracy_model <- which.max(apply(cv10.model.comparison, 2, mean))

  if(highest_accuracy_model == 1) {
    rf.fit <- randomForest(new_formula, data = train_data_small, subset =
↪   train_indices_cv)
    double.cv.predictions[validation_indices_cv] <- predict(rf.fit, validation_data_cv,
↪   type="class")
  } else {
    svm.fit <- svm(new_formula, data = train_data_small, kernel="linear", scale=FALSE,
↪   type="C-classification")
    double.cv.predictions[validation_indices_cv] <- predict(svm.fit, validation_data_cv,
↪   type="class")
  }
}

printf("\n")
end_time <- Sys.time()

end_time - start_time
```

Calculation time: 5.3 minutes

5.5.1 Results of double CV-10:

```
##            actuals
## predictions   No  Yes  Sum
##         1   4763  656 5419
##         2    271  742 1013
##         Sum 5034 1398 6432
```

```
##
## Accuracy of double CV-10:  0.8558769
```

Double CV-10 adds another layer of cross-validation, which gives an improved approximation of the model's prediction accuracy on the test data.

6 Parameter tuning for SVM and comparison to other models

3 things happen below:

1. We perform grid search CV-10 on 10 randomly chosen cost parameters for SVM.

- Random forest was already tuned during the feature selection stage.

2. We will also perform CV-10 on three other models: random forest with mtry = p, neural net, and LDA (linear discriminant analysis)... If all 4 models in this analysis (SVM, random forest, neural net and LDA) have similar test set prediction accuracies - then its possible that we have peaked in prediction accuracy with this set of predictors.
3. Parallel processing is used to improve training time.

Note: After complete cross-validation of each model, Caret's *train* function returns the final model which is trained on the full training set.

```
set.seed(6)

start_time <- Sys.time()

# define cores and register parallel computation operation
cl <- makePSOCKcluster(detectCores())
registerDoParallel(cl)

# set grid search hyperparameters
max_hyperparams = 10

# define additional hyperparameters
tr.Control <- trainControl(method='cv',
                          number=10,
                          allowParallel = T,
                          verboseIter = F,
                          classProbs = TRUE)

# define grids for methods
mlp.grid <- expand.grid(layer1 = 10, layer2 = 7, layer3 = 2)
svm.grid <- expand.grid(C = sort(c(sample(seq(.01, 1, .1), max_hyperparams/2),
↪  sample(seq(1, 20, 2), max_hyperparams/2))))
bagging.grid <- expand.grid(mtry = length(predictors))

# train models
svm.fit <- train(new_formula, data = train_data_small, method = "svmLinear",
↪  trControl=tr.Control, tuneGrid = svm.grid)
bagging.fit <- train(new_formula, data = train_data_small, method = "parRF",
↪  trControl=tr.Control, tuneGrid = bagging.grid)
# mlp is the same as nnet but allows multiple hidden layers
nn.fit <- train(new_formula, data = train_data_small, method = "mlpML", trControl=
↪  tr.Control, tuneGrid = mlp.grid, metric="ROC")
lda.fit <- train(new_formula, data = train_data_small, method = "lda", trControl=
↪  tr.Control)

stopCluster(cl)

end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 1.409152 mins
```

Calculation time: 1.4 minutes

6.1 Predictions on the (compact) test data with the final model of each method

```
## Accuracy of SVM model with C =  0.81 : 0.8439055
```

```
## 
## Accuracy of bagging model: 0.835199
```

```
## 
## Accuracy of neural net with 3 layers: 0.8364428
```

```
## 
## Accuracy of LDA: 0.8364428
```

SVM produces the best model of those trained with 84.39% accuracy on the compact test set. A shallow neural net with 3 layers and LDA come in 2nd and 3rd, with scores of 83.64% and 83.64%, respectively. Bagging arrives last at 83.5%

6.2 Best model predictions on test data

```
best_model_fit = svm(new_formula, data = train_data, kernel = "linear", scale=FALSE,
↪  type="C-classification", cost = svm.fit$bestTune)
test_predictions <- predict(best_model_fit, test_data)
test_accuracy <- get_accuracy(test_predictions, test_data$raintomorrow)
```

```
##            actuals
## predictions    No   Yes   Sum
##         No   7949  1178  9127
##         Yes   390  1203  1593
##         Sum  8339  2381 10720
```

```
cat("\nBest model accuracy on test data:", test_accuracy)
```

```
## 
## Best model accuracy on test data: 0.8537313
```

6.3 Best model predictions on full data

```
all_data <- rbind(train_data, test_data)
final_fit = svm(new_formula, data = all_data, kernel = "linear", scale=FALSE,
↪  type="C-classification", cost = svm.fit$bestTune)
final_predictions <- predict(final_fit, all_data)
full_accuracy <- get_accuracy(final_predictions, all_data$raintomorrow)
```

```
##            actuals
## predictions    No   Yes   Sum
##         No  39806  5838 45644
##         Yes  2024  5931  7955
##         Sum 41830 11769 53599
```

```
cat("\nAccuracy on all data:", full_accuracy)
```

```
##
## Accuracy on all data: 0.8533182
```

7 Conclusion

The methodology and process used in this analysis results in a prediction accuracy of ~85.4% on the test data of 10720 observations (~25% of the training data size of 42879). I believe that a deep neural network would probably be the best model to achieve >= 90% prediction accuracy. In order to attain at least 90% rain prediction accuracy without a deep neural net, this current dataset would have to be upgraded with more valuable features, such as detailed forecasts, oceanic readings, atmospheric readings at different altitudes, and intra-daily (at least hourly) weather changes.

Furthermore, as tested in this analysis through the utilization of lags of *RainToday* and percent of prior n day's rain, autoregressive time series analysis for rain prediction is not useful for rain prediction, even in addition to existing atmospheric predictors. A possibly useful autoregressive approach would be spatial autoregressive classification.

8 References

[1] p. 378, An Introduction to Statistical Learning