Arrest Classification

Imports

```r
# Load required libraries
library(tidyverse)
library(lubridate)
library(randomForest)
library(e1071)
library(xtable)

# Load package functions
devtools::load_all("../chigcrim/")

# Use seed to ensure reproducibility
set.seed(1)
```

Introduction

In this section, the binary classification task of predicting if a suspect is arrested is considered. Three models will be investigated, logistic regression, support vector machines and random forest classifiers.

Assessing performance

To assess performance, three metrics will be used, the overall accuracy, the sensitivity and the specificity. These are defined as follows:

- Overall accuracy: The proportion of correct predictions.
- Sensitivity: Proportion of positive observations correctly predicted.
- Specificity: Proportion of negative observations correctly predicted.

Note that as logistic regression is a probabilistic classifier, the results are rounded to yield predictions that can be assessed with the metrics above to facilitate the comparison with the support vector machine results.

```r
# List of metrics used to evaluate classification models
# These are used by cv_eval function
metrics <- list(accuracy = classification_accuracy,
                sensitivity = classification_sensitivity,
                specificity = classification_specificity)
```

For each classifier, we use repeated k-fold cross validation. Specifically, 5-fold cross validation is repeated three times (run in parallel), and the metrics are averaged across the folds.

Feature selection

Due to computational limitations, only the 2019 data will be considered. Below, the feature selection choices are outlined. These decisions involved incorporating information from the exploratory data analysis (EDA), as well as considering what is computationally feasible.

- Encode the date as the day in the year (1-365).
- Encode time of day as a float 0-24 (this looks good in the EDA).
- Drop `year` as these are all 2019.
- We will drop the `ids`, as it contains unique values.
- We drop `case_number`, as it contains almost all unique values.
- Drop `primary_type`, `description` and `iucr` code. Keep `fbi_code` as an indicator of crime type.
- Assume `updated_on` is not informative (see EDA).
- `latitude` and `longitude` are dropped. `x_coordinates` and `y_coordinates` are kept. Note that these coordinates likely not particularly useful for linear classifiers, but should be useful for non-linear methods.
- Community areas will be kept, but other areas (`district`, `beat`, `ward` and `block`) are dropped.
- NAs will be dropped (see EDA).
- Particularly rare factors will be grouped into a variable other (see `?otherise`).

```r
# Load 2019 data to build classification models
df <- load_data(year = 2019, strings_as_factors = FALSE)

# Convert rareuly used fbi categories to other
df$fbi_code <- otherise(df$fbi_code, 500)
```

```
## [1] "7 out of 26 categories were converted to OTHER corresponding to 0.512076404563392% of observati
```

```r
df$location_description <- otherise(df$location_description, 1000)
```

```
## [1] "126 out of 155 categories were converted to OTHER corresponding to 10.8184777434858% of observa
```

```r
# List of features for removal, EDA showed not necessary
remove_features <- c("id", "year", "case_number", "primary_type", "description",
                     "iucr", "updated_on", "latitude", "longitude",
                     "date", "district", "beat", "ward", "block")

# Add more time features
df <- df %>%
  mutate(day = yday(df$date),
         time = hour(df$date) + minute(df$date)/60) %>%
  select(-all_of(remove_features)) %>%
  filter(complete.cases(df))

# Convert to factors
```

```
df <- df %>%
  mutate(location_description = as.factor(location_description),
         fbi_code = as.factor(fbi_code),
         community_area = as.factor(community_area))

head(df)
```

```
## # A tibble: 6 x 9
##   location_descri~ arrest domestic community_area fbi_code x_coordinate
##   <fct>            <lgl>  <lgl>    <fct>          <fct>           <int>
## 1 RESTAURANT       FALSE  FALSE    14             14            1153943
## 2 RESIDENCE        FALSE  FALSE    44             14            1182085
## 3 HOTEL/MOTEL      FALSE  FALSE    8              04B           1175159
## 4 ALLEY            TRUE   FALSE    23             15            1151958
## 5 APARTMENT        FALSE  TRUE     29             08B           1152589
## 6 RESIDENCE        FALSE  FALSE    30             14            1155950
## # ... with 3 more variables: y_coordinate <int>, day <dbl>, time <dbl>
```

Logistic Regression

We will first consider a logistic regression classifier. This is a linear classifier. It calls the function `optim()` internally to minimize the cross-entropy/log-loss function using gradient based optimisation methods (here BFGS).

```
# Split into data matrix and response vector
X <- df %>% select(-arrest)
y <- df$arrest

# Initialise lr class
lr <- LogisticRegression$new(solver = "BFGS",
                             control = list(maxit=1000, reltol = 1e-4),
                             round_y_hat = TRUE)

# Find repeated cv error
lr_results <- kfold_cv(lr, X, y, metrics, k = 5, n_reps = 5,
                       parallel = TRUE, n_threads = 5)


as_tibble(lr_results)
# Export results for use in report
xtable(as_tibble(lr_results), digits = 4)
```

```
## # A tibble: 5 x 3
##   accuracy sensitivity specificity
##      <dbl>       <dbl>       <dbl>
## 1    0.861       0.435       0.979
## 2    0.861       0.435       0.979
## 3    0.861       0.435       0.979
## 4    0.861       0.434       0.979
## 5    0.861       0.435       0.979
```

```
## % latex table generated in R 4.0.3 by xtable 1.8-4 package
## % Sun Jan 24 16:04:40 2021
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrr}
##   \hline
##  & accuracy & sensitivity & specificity \\
##   \hline
## 1 & 0.8614 & 0.4347 & 0.9788 \\
##   2 & 0.8613 & 0.4351 & 0.9786 \\
##   3 & 0.8613 & 0.4345 & 0.9787 \\
##   4 & 0.8613 & 0.4341 & 0.9789 \\
##   5 & 0.8612 & 0.4346 & 0.9786 \\
##    \hline
## \end{tabular}
## \end{table}
```

Support Vector Machine

As the support vector machine allows use of a kernel function, it should be able to capture non-linear relationships in the original feature space (given an appropriate kernel). Here, a radial basis function kernel is used. This implementation is a wrapper of the SVM package `e1071`. Unfortunately, support vector machines do not scale well to large data sets, so here we will only use 30000 rows.

We first need to find the optimal hyperparameters which we do by conducting a grid search. We use a smaller data set here as the models need to be fitted many times.

```r
idxs <- sample(1:nrow(df), 20000)
X_tune <- X[idxs, ]
y_tune <- y[idxs]

svm_tuned <- tune(e1071::svm, y ~ ., data = cbind(X_tune, y_tune),
          ranges = list(cost = 2^(-2:5),gamma = 2^(-15:-4)),
          tunecontrol = tune.control(nrepeat = 3,
          sampling = "cross", cross = 5),
          kernel = "radial", type = "C-classification")

svm_tuned
```

```
##
## Parameter tuning of 'e1071::svm':
##
## - sampling method: 5-fold cross validation
##
## - best parameters:
##  cost   gamma
##    16 0.03125
##
## - best performance: 0.128
```

With these hyperparameters we then find the 5x repeated cross-validated error on a subset with 30,000 rows.

```r
idxs <- sample(1:nrow(df), 30000)
X <- X[idxs, ]
y <- y[idxs]
svm <- SupportVectorMachine$new(kernel = "radial")

svm_results <- kfold_cv(svm, X, y, metrics, 5, 5, parallel = FALSE,
                        n_threads = 5, gamma = 2^-5, cost = 2^4)




# Export results for use in report
as_tibble(svm_results)
xtable(as_tibble(svm_results), digits = 4)
```

```
## # A tibble: 5 x 3
##   accuracy sensitivity specificity
##      <dbl>       <dbl>       <dbl>
## 1    0.862       0.408       0.987
## 2    0.863       0.409       0.987
## 3    0.863       0.409       0.987
## 4    0.862       0.407       0.987
## 5    0.862       0.405       0.987
```

```
## % latex table generated in R 4.0.3 by xtable 1.8-4 package
## % Sun Jan 24 16:04:40 2021
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrr}
##   \hline
##  & accuracy & sensitivity & specificity \\
##   \hline
## 1 & 0.8624 & 0.4076 & 0.9870 \\
##   2 & 0.8627 & 0.4091 & 0.9870 \\
##   3 & 0.8626 & 0.4087 & 0.9870 \\
##   4 & 0.8622 & 0.4067 & 0.9870 \\
##   5 & 0.8619 & 0.4052 & 0.9870 \\
##    \hline
## \end{tabular}
## \end{table}
```

Random Forest

The random forest trains multiple decision trees using a subset of features to create each tree. Decision trees are an ordered set of rules defined on the feature values that attempt to split the data into classes. They have a tendency to overfit, the random forest solves this by growing multiple trees on different features within the dataset. In the classification case a prediction is made by predicting from each decision tree separately and finding the most common predicted class. They are a black-box method but can be useful for prediction. The R6 class used here uses the `randomForest` package internally. Due to the high computational cost involved in fitting random forest models we again use a randomly selected subset of the data (20000 rows) as we did for the SVM.

```r
X <- select(X, - "community_area")  # Can't handle feature with so many levels.

# Initialise random forest class
rf <- RandomForest$new()

rf_results <- kfold_cv(rf, X, y, metrics, 5, 5, parallel = TRUE, n_threads = 5)


# Export results for use in report
as_tibble(rf_results)
xtable(as_tibble(rf_results), digits = 4)
```

```
## # A tibble: 5 x 3
##   accuracy sensitivity specificity
##      <dbl>       <dbl>       <dbl>
## 1    0.876       0.526       0.972
## 2    0.878       0.529       0.974
## 3    0.876       0.530       0.971
## 4    0.877       0.529       0.973
## 5    0.876       0.526       0.972
```

```
## % latex table generated in R 4.0.3 by xtable 1.8-4 package
## % Sun Jan 24 16:04:40 2021
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrr}
##   \hline
##  & accuracy & sensitivity & specificity \\
##   \hline
## 1 & 0.8762 & 0.5258 & 0.9722 \\
##   2 & 0.8780 & 0.5293 & 0.9736 \\
##   3 & 0.8762 & 0.5299 & 0.9712 \\
##   4 & 0.8775 & 0.5293 & 0.9729 \\
##   5 & 0.8761 & 0.5259 & 0.9721 \\
##    \hline
## \end{tabular}
## \end{table}
```

Comparison of models

The models all performed very similarly. Unsurprisingly, the support vector machine and the random forest classifier performed better, due to their ability to utilise non-linear relationships. It is worth noting that this difference is surprisingly small, although this may be due to the fact that logistic regression could be trained using more data. Alternatively, it could suggest that encoding the community areas as factors was sufficient to capture much of the spatial aspect of the crimes in logistic regression.

Logistic regression had higher sensitivity than the support vector machine, so could be a preferable option if false negatives are particularly detrimental, although it is worth noting that both support vector machines and logistic regression can be adjusted to reflect unbalanced costs associated with incorrect predictions.

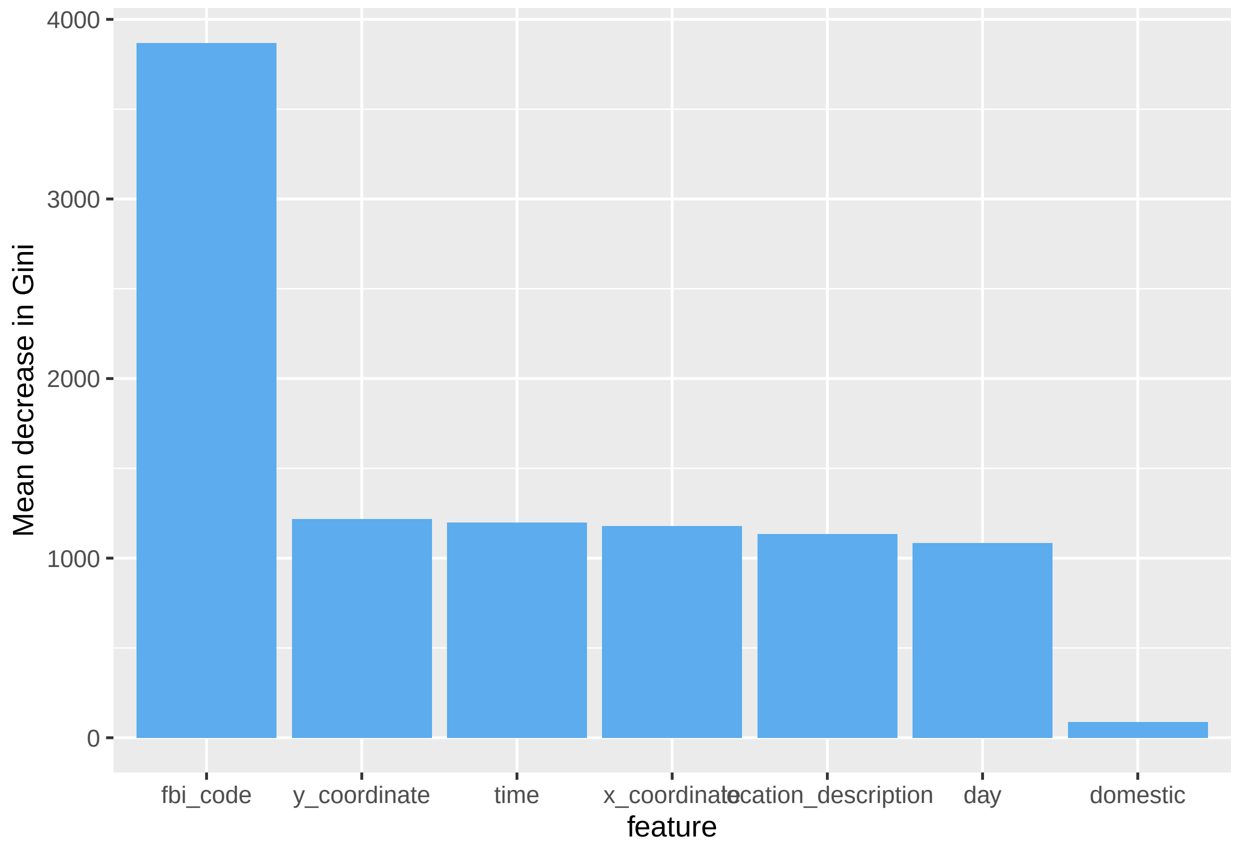
Being an ensemble model, the random forest unsurprisingly performed the best based on all three metrics. It is likely that the random forest could be improved by training on a larger subset of the data and optimising the hyperparameters. The downside of this is that as a black-box method is does not provide any inference as to how the features are used within the model. We are at least able to look at how many times each feature was chosen as a splitting point in all of the trees providing some inference as to the importance of each of the features.

```r
# Fit model
rf$fit(X, y)

# get importance of features from model
imp <- importance(rf$fitted_model)
imp <- tibble(feature = row.names(imp), mean_decrease_gini = as.vector(imp))

imp <- imp %>%
  arrange(desc(mean_decrease_gini)) %>%
  mutate(feature = factor(feature, levels = .$feature))

# Plot importance
imp_plot <- ggplot(aes(x = feature, y = mean_decrease_gini), data = imp) + geom_col(fill = "steelblue2")
imp_plot
```



```r
ggsave("../plots/rf_imp.pdf", plot = imp_plot, width = 7.5, height = 5)
```

We see the most important feature was `fbi_code`, this is somewhat expected as it determines what type of crime occurred and this would intuitively have a large affect in whether someone was arrested. None of the models achieve an accuracy of more than 90%. The availability of more data, such as the demographic background of the suspect, would likely allow for more accurate predictions.