Compass TB1 Group Project: Lab Notebook

Euan Enticott, Daniel Ward, Shannon Williams

25/01/2021

Introduction

In this lab notebook we present the analyses undertaken as part of the Compass TB1 group project. We focus on a large, open source dataset provided by the City of Chicago consisting of reported incidents of crime from 2001 to present day, as reported by the Chicago Police Department. The dataset can be obtained from the website https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2.

The notebook will be broken up into four sections:

1. Workflow and Computational Techniques: A summary of the approach used in the development of the **chigcrim** package.
2. Classification: Predicting whether a reported crime led to the arrest of a suspect. We consider three different discriminative classifiers: logistic regression, support vector machines, and random forests.
3. Prediction: Predicting the number of reported crimes which occur in a given region over a specified time period (of a given type, if desired), through interpolation. We consider kernel ridge regression and generalised additive models (GAMs).
4. Forecasting: Forecasting the number of crimes which occur in a given region over a specified time period (of a given type, if desired), based on previous time-stamped data. Again, we consider the use of GAMs.

Workflow and Computational Techniques

In the development of the package for this project, we utilised many computational techniques, that are outlined below.

Object-Oriented Programming

We decided to use an object oriented approach to constructing models. Specifically, we made use of the `R6Class` from the **R6** package. `R6` objects are mutable and use an object oriented approach similar to most programming languages (unlike the more basic `S3` and `S4` classes). `R6` classes are very similar to reference classes, although generally `R6` classes are preferred. This is for a variety of reasons, including the fact that `R6` classes are much faster, and they handle fields in a neater way by putting them in a separate environment. More information on this can be found here. When building models, a model could be fitted using `model$fit(X, y)`, and predictions made with `model$predict(X)`. It is worth noting that this use of `fit` and `predict` methods in object-oriented machine learning packages is used elsewhere, particularly in the **Python** package **scikit-learn**. The key advantage of this approach is that it abstracts away the underlying complexities, and creates a consistent structure that makes it easier to compare different models.

Parallel Programming

Parallel programming was used to speed up slow code, utilising the package **doParallel** which calls the packages **parallel** and **foreach**. In particular, we implemented a parallel cross-validation function `kfold_cv`, which facilitated more accurate and faster assessment of model performance.

Documentation

In order to document functions in the package, we made use of **roxygen2**. This package automatically creates the `.Rd` documentation files for functions (or classes), from comments added above the function definitions. As well as being efficient, this allows the code and documentation to coexist, meaning it is easier to remember to update the documentation.

API Querying

The Chicago Crime dataset is updated on a daily basis and, at the time of writing, contains over 7 million rows of data. For each reported incident, a total of 22 features are recorded (for a full list of features and their descriptions see the City of Chicago website). We elected not to download the dataset in full and include this in the package as it would consume a significant amount of storage, and quickly become out of date. Instead, we utilised the **RSocrata** package which allows easy interaction with the online open data portal. The wrapper `load_data` allows the end user to directly download the dataset into the R environment, via the function `read.socrata`. `load_data` provides functionality for filtering by year, limiting the number of data points received, and omitting missing/NA values.

Testing and Continuous Integration

For tests, we used the **testthat** package. This simply creates a new directory `./tests/testthat/` in which the user can define tests for the package functions. Although the tests can be run alone, generally, we made use of the package **rcmdcheck**. This contains the function `rcmdcheck`, which not only runs the tests, but also carries out a more sophisticated check. Some examples of what is checked are listed below:

- Checks the package installs correctly.
- Checks for missing documentation.
- Automatically runs examples in documentation, to check they run successfully.
- Checks for undefined variables.
- Checks for missing dependencies.

In order to make sure that tests and checks were run on a regular basis, continuous integration was set up using github actions. This runs `rcmdcheck()` for each pull request and push to the main branch. This limits the possibility of unintentionally introducing breaking changes.

Classification

```
# Load required packages
library(chigcrim)

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

In this section we consider the binary classification task of predicting whether a suspect is arrested, given data on the reported crime. 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: The proportion of positive observations correctly predicted.
- Specificity: The proportion of negative observations correctly predicted.

These metrics can be computed using the functions `classification_accuracy`, `classification_sensitivity`, and `classification_specifity`, respectively, from the **chigcrim** package.

Note that as logistic regression is a probabilistic classifier whereas support vector machines and random forests are not. As such, the output from logistic regression is a probability in $[0, 1]$. In our analysis, these probabilities are rounded to yield predictions that can be assessed with the metrics above to facilitate the comparison with the results of the other two classifiers.

```
# List of metrics used to evaluate classification models
metrics <- list(accuracy = classification_accuracy,
                sensitivity = classification_sensitivity,
                specificity = classification_specificity)
```

For each classifier, we will use repeated k-fold cross validation to obtain mean values of the metrics. Specifically, 5-fold cross validation is repeated five times (run in parallel using the function `kfold_cv`), and the metrics are averaged across the folds and repeats.

Feature selection

Due to the time and computational limitations involved, we shall train the models on a subset of the data from the year 2019.

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

```
print(as.data.frame(head(df)))
```

```
##         id case_number                date                  block iucr
## 1 11552577    JC100040 2019-01-01 00:31:00     032XX W LAWRENCE AVE 1310
## 2 11552587    JC100034 2019-01-01 00:05:00          006XX E 83RD PL 1310
## 3 11552596    JC100045 2019-01-01 00:03:00         001XX W HURON ST 0430
## 4 11552605    JC100030 2019-01-01 00:01:00    004XX N MONTICELLO AVE 143A
## 5 11552609    JC100028 2019-01-01 00:05:00 013XX S CENTRAL PARK AVE 0486
## 6 11552610    JC100044 2019-01-01 00:02:00          031XX W 25TH ST 1310
##        primary_type                   description location_description arrest
## 1   CRIMINAL DAMAGE                   TO PROPERTY           RESTAURANT  FALSE
## 2   CRIMINAL DAMAGE                   TO PROPERTY            RESIDENCE  FALSE
## 3           BATTERY AGGRAVATED: OTHER DANG WEAPON          HOTEL/MOTEL  FALSE
## 4 WEAPONS VIOLATION      UNLAWFUL POSS OF HANDGUN                ALLEY   TRUE
## 5           BATTERY       DOMESTIC BATTERY SIMPLE            APARTMENT  FALSE
## 6   CRIMINAL DAMAGE                   TO PROPERTY            RESIDENCE  FALSE
##   domestic beat district ward community_area fbi_code x_coordinate y_coordinate
## 1    FALSE 1713       17   33             14       14      1153943      1931709
## 2    FALSE  632        6    6             44       14      1182085      1849762
## 3    FALSE 1832       18   42              8      04B      1175159      1905043
## 4    FALSE 1122       11   27             23       15      1151958      1902815
## 5     TRUE 1011       10   24             29      08B      1152589      1893426
## 6    FALSE 1033       10   12             30       14      1155950      1887234
##   year          updated_on latitude longitude
## 1 2019 2019-01-10 15:16:50 41.96844 -87.70934
## 2 2019 2019-01-10 15:16:50 41.74297 -87.60841
## 3 2019 2019-01-10 15:16:50 41.89482 -87.63213
```

```
## 4 2019 2019-01-10 15:16:50 41.88920 -87.71740
## 5 2019 2019-01-10 15:16:50 41.86342 -87.71533
## 6 2019 2019-01-10 15:16:50 41.84636 -87.70316
```

Below we outline our feature selection and data pre-processing choices. These decisions involved incorporating information from conducting exploratory data analysis (EDA), as well as considering what is computationally feasible.

- Encode the date as the day of the year (an integer between 1-365, or 1-366 for leap years).
- Encode time of day as a floating point in 0-24, as this showed promising results in our EDA.
- Drop `year` as we are only using data from 2019.
- Drop `id` as this is the unique key for each data point.
- We drop `case_number`, as the values are almost all unique and are not informative.
- Drop `primary_type`, `description` and `iucr` code. Keep `fbi_code` as an indicator of crime type.
- Drop `updated_on` as this only refers to when the data was last updated and is uninformative.
- Drop `latitude` and `longitude`, and keep`x_coordinate` and `y_coordinate`. Note that these coordinates likely not particularly useful for linear classifiers, but should be useful for non-linear methods.
- Keep `community_area`, but other areas (`district`, `beat`, `ward` and `block`) are dropped.
- Omit all `NA` values.
- Particularly rare factors will be grouped into a variable other (see `?otherise`).

```r
# Convert rarely 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 0.512072473225596% of observati
```

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

```
## [1] "126 out of 155 categories were converted to OTHER corresponding to 10.8183946873441% of observa
```

```r
# List of features for removal
remove_features <- c("id", "year", "case_number", "primary_type", "description",
                     "iucr", "updated_on", "latitude", "longitude",
                     "date", "district", "beat", "ward", "block")

# Add more time features, remove missing values and convert columns to factor
df %<>% mutate(day = yday(df$date),
               time = hour(df$date) + minute(df$date) / 60) %>%
  select(-all_of(remove_features)) %>%
  filter(complete.cases(df)) %>%
  mutate(location_description = as.factor(location_description),
         fbi_code = as.factor(fbi_code),
         community_area = as.factor(community_area))

print(as.data.frame(head(df)))
```

```
##   location_description arrest domestic community_area fbi_code x_coordinate
## 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
##   y_coordinate day       time
## 1      1931709   1 0.51666667
## 2      1849762   1 0.08333333
## 3      1905043   1 0.05000000
## 4      1902815   1 0.01666667
## 5      1893426   1 0.08333333
## 6      1887234   1 0.03333333
```

Logistic Regression

We will first consider a logistic regression classifier, which is a linear classifier for binary data. We use our R6 class `LogisticRegression` which has methods `fit` and `predict`. `fit` calls the function `optim()` internally to minimize the cross-entropy/log-loss function using gradient-based optimisation methods (either BFGS, L-BFGS-B or CG); in our analysis we use BFGS as we don't run into memory issues.

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

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

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

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

```
## % latex table generated in R 4.0.3 by xtable 1.8-4 package
## % Sun Jan 24 16:58:03 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 Machines

As the support vector machine (SVM) allows use of a kernel function, it should be able to capture non-linear relationships in the original feature space (given an appropriate kernel). As the relationship between the features and the class is unknown, a radial basis function kernel is used. The R6 class `SupportVectorMachine`, again with `fit` and `predict` methods, is a wrapper of the function `svm` from the dedicated SVM package **e1071**. Unfortunately, support vector machines do not scale well to large data sets, so here we will only use

30,000 rows.

```r
# Subset the data randomly
index <- sample(1:nrow(df), 30000)
X_subset <- X[index, ]
y_subset <- y[index]
```

In order to obtain optimal values for `cost` and `gamma` to be provided to `svm` for training the SVM, we utilise the built-in generic function `tune` from **e1071** which performs a grid search over given parameter ranges.

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

```r
svm_tuned
```

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

We obtain optimal values of `2^4` for `cost` and `2^-5` for `gamma`, which we use to train our SVM.

```r
# Initialise new SupportVectorMachine object
svm <- SupportVectorMachine$new(kernel = "radial")
```

```r
# Run repeated 5-fold cross-validation to evaluate performance
svm_results <- kfold_cv(svm, X_subset, y_subset, metrics, k = 5,
                        n_reps = 5, parallel = FALSE,
                        n_threads = 5, gamma = 2^-5, cost = 2^4)
```

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

```
## % latex table generated in R 4.0.3 by xtable 1.8-4 package
## % Sun Jan 24 16:45:30 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. As decision trees have a tendency to overfit, the random forest attempts to overcome 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. This is a black-box method but has the potential for good predictive performance.

Our R6 class `RandomForest` is a wrapper for the function `randomForest` from the **randomForest** package. Due to the high computational cost involved in fitting random forest models we again use a smaller subset of the data. Here we use the same dataset used to fit the SVM, with the factor `community_area` removed as the algorithm for fitting the model cannot handle such a large number of factors.

```r
X_subset <- select(X_subset, - "community_area")

# Initialise new RandomForest object
rf <- RandomForest$new()
```

```r
# Run repeated 5-fold cross-validation to evaluate performance
rf_results <- kfold_cv(rf, X_subset, y_subset, metrics, k = 5,
                       n_reps = 5, parallel = TRUE, n_threads = 5)
```

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

```
## % latex table generated in R 4.0.3 by xtable 1.8-4 package
## % Sun Jan 24 16:45:59 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}
```

Model Comparisons

The models all performed very similarly. Unsurprisingly, the support vector machine and the random forest classifier performed better, due to their ability to capture non-linear relationships. It is worth noting that this difference is surprisingly small, although this could be related to the fact that the logistic regression model was trained on a larger dataset. Alternatively, the SVM results 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, and thus could be viewed as a preferable option if false negatives are particularly detrimental. It is worth noting however that both support vector machines and logistic regression can be adjusted to reflect unbalanced costs associated with incorrect predictions.

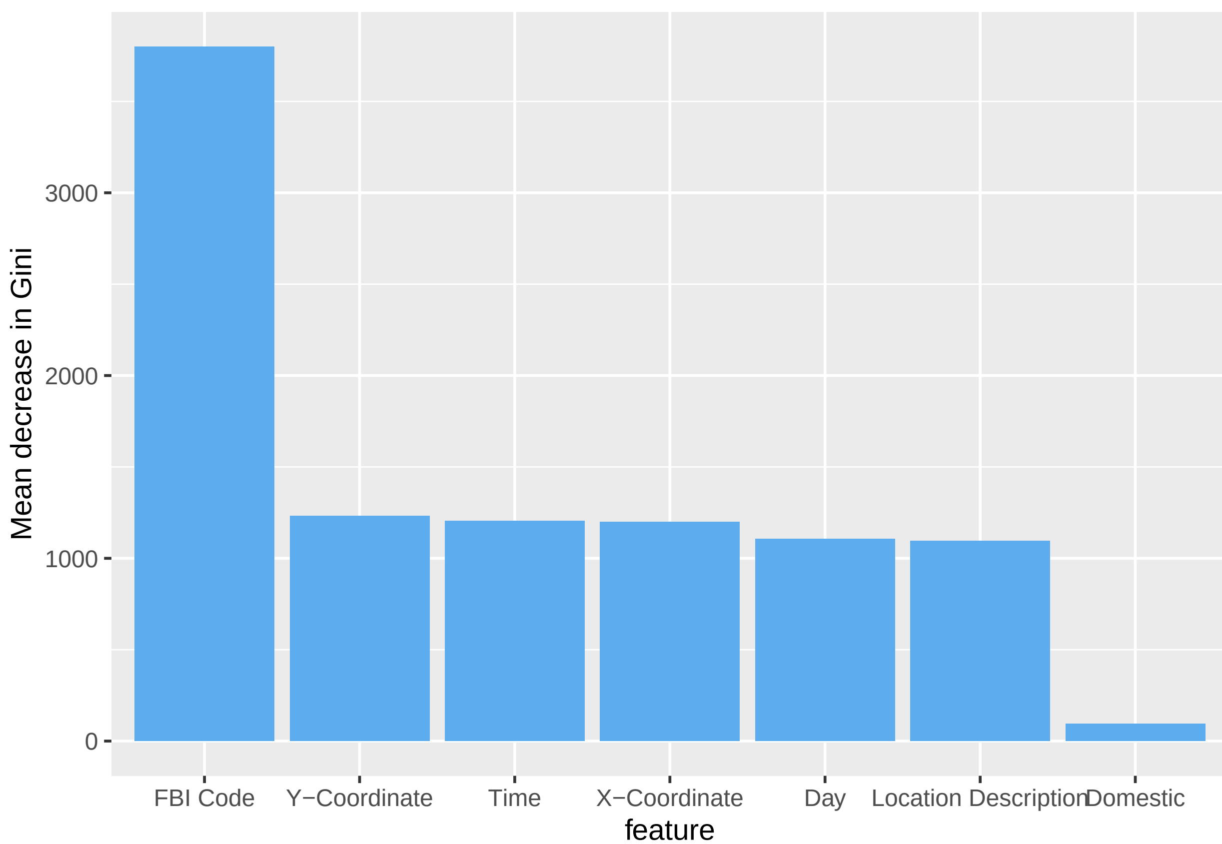
Being an ensemble model, the random forest unsurprisingly performed the best based on all three metrics. It is likely that the random forest could be improved by training on a larger subset of the data and optimising the hyperparameters. The downside of using a random forest is that, as a black-box method, it 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 us with some information as to the importance of each of the features.

```r
# Fit model with optimal hyper parameters
rf$fit(X_subset, y_subset)

# Get importance of features from model
imp <- randomForest::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") + ylab("Mean decrease in Gini") +
  scale_x_discrete(labels = c("FBI Code", "Y-Coordinate", "Time", "X-Coordinate",
                              "Day", "Location Description", "Domestic"))
imp_plot
```



The plot suggests that the most "important" feature is `fbi_code`, which is somewhat unsurprising as this tells us what type of crime was reported. This would intuitively have a significant impact on whether the incident report led to a suspect being arrested. None of the models achieve an accuracy of more than 90%. The availability of more data, for example the demographic background of the suspect and whether they have been arrested previously, would likely allow for more accurate predictions.

Prediction

In this section we consider the regression tesk of predicting the number of reported incidents of crime in a given region over a specified time period. We will look at a kernel ridge regression model before moving on to generalised additive models.

```
# Load required packages
library(chigcrim)
library(rgdal)
library(sf)
library(ggmap)
# Use seed to ensure reproducibility
set.seed(1)
```

Assessing Performance

To assess performance, two metrics will be used: the root mean squared error (RMSE) and R-squared (R^2). These are defined as follows:

- RMSE:
- R-squared:

These metrics can be computed using the functions `rmse_loss` and `r_squared`, respectively, from the **chigcrim** package.

```
# List of metrics used to evaluate regression models
metrics <- list(rmse = rmse_loss, r2 = r_squared)
```

Repeated k-fold cross validation is used, and hyperparameters are chosen based on the root mean square error metric. Specifically, we will use 5 repetitions of 5-fold cross validation at each parameter value (ran in parallel).

Kernel Ridge Regression

The kernel ridge regression implementation includes a linear kernel (equivalent to ridge regression), a polynomial kernel, and a radial basis function kernel. As the relationship is complex and non-linear, only the radial basis function feature transform will be considered here, which implicitly induces an extremely flexible infinite dimensional feature transform.

Weekly predictions

We first consider a simple, single-dimensional model, using only the number of crimes per week as the predictor.

```
df <- load_data(select = "date")
```

```
df %<>%
  mutate(date = date(date), week = week(date), year = year(date)) %>%
  filter(week < 53) %>%
  group_by(week, year) %>%
  mutate(week_start = min(date)) %>%
  ungroup() %>%
```

```r
  count(week, week_start, year) %>%
  arrange(year, week) %>%
  mutate(cumulative_week = 1:nrow(.))
print(as.data.frame(head(df)))
```

```
##   week week_start year    n cumulative_week
## 1    1 2001-01-01 2001 8942               1
## 2    2 2001-01-08 2001 8589               2
## 3    3 2001-01-15 2001 8704               3
## 4    4 2001-01-22 2001 8300               4
## 5    5 2001-01-29 2001 8473               5
## 6    6 2001-02-05 2001 8418               6
```

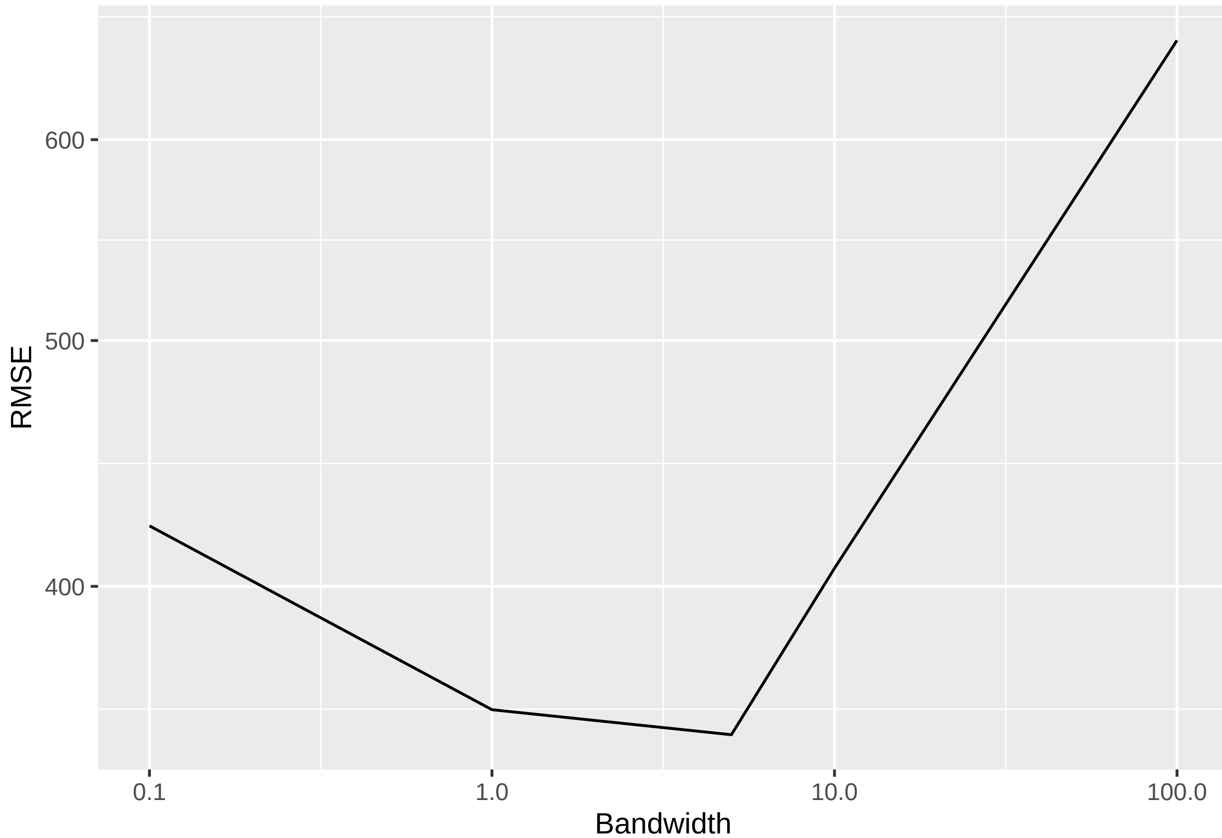
We shall use a grid search with 5-fold cross validation to choose the bandwidth hyperparameter for the radial basis function kernel.

```r
X <- as.matrix(df$cumulative_week)
X <- scale(X)
y <- df$n
```

```r
# Candidate values for the bandwidth of the radial basis function
bandwidth <- c(0.1, 1, 5, 10, 100)
# Initisalise vector of RMSE
rmse_vec <- c()
# Conduct a grid search with 5-fold CV
for (i in 1:length(bandwidth)){
  kr = KernelRidge$new("rbf", lambda = 0.001, bandwidth[i])
  cv_results <- kfold_cv(kr, X, y, metrics, 5, n_reps = 5,
                         parallel = TRUE, n_threads = 5)
  mean_rmse <- mean(cv_results$rmse)
  rmse_vec <- c(rmse_vec, mean_rmse)
  # If mean RMSE is better than for the last best model clone and treat as best
  if (mean_rmse == min(rmse_vec)){
    # Clone the model and save
    kr_best <- kr$clone(deep = TRUE)
    # Update the best results
    cv_results_best <- cv_results
  }
}

rbf_results <- data.frame(bandwidth, rmse = rmse_vec)

bandwidth_plot <- ggplot(rbf_results) +
  geom_line(aes(x = bandwidth, y = rmse)) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Bandwidth") + ylab("RMSE")
bandwidth_plot
```



We can see that a bandwidth parameter value of 5 gave the lowest root mean square error. We can look at the results for the three repeats of 5-fold cross-validation for this model:

```
xtable::xtable(as_tibble(cv_results_best), digits = 4)
```

```
## % latex table generated in R 4.0.3 by xtable 1.8-4 package
## % Sun Jan 24 16:58:21 2021
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrr}
##   \hline
##  & rmse & r2 \\
##   \hline
## 1 & 353.3775 & 0.9627 \\
##   2 & 347.7463 & 0.9640 \\
##   3 & 351.1616 & 0.9635 \\
##   4 & 346.9327 & 0.9646 \\
##   5 & 348.7730 & 0.9642 \\
##    \hline
## \end{tabular}
## \end{table}
```

The model performs very well, explaining 96.38 % of the variance on average.

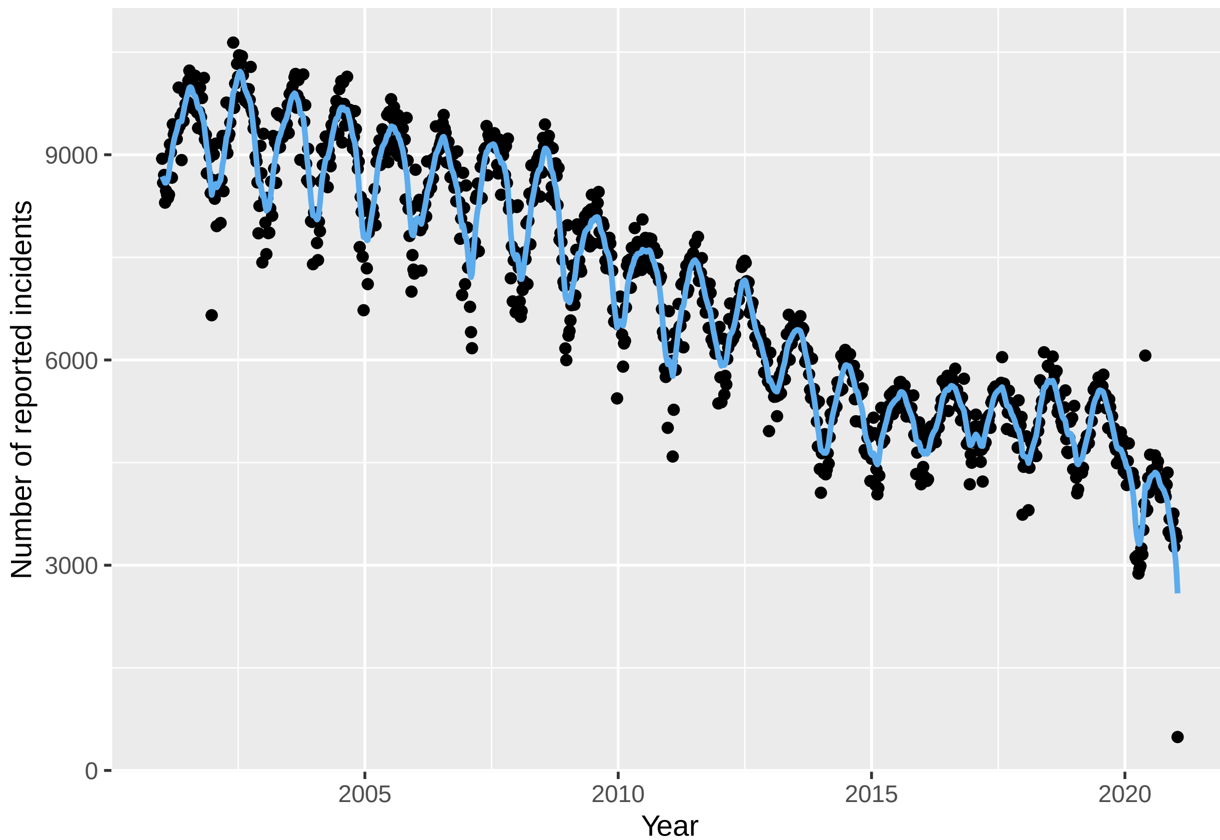
As this is single-dimensional, we can plot the prediction function for the best model.

```
# Fit the final kernel ridge regression model
kr_best$fit(X, y)
```

```r
# Obtain predictions on the dataset
y_hat <- kr_best$predict(X)

# Plot output
df$y_hat <- y_hat

week_plot <- ggplot(df) +
  geom_point(aes(x = week_start, y = n)) +
  geom_line(aes(x = week_start, y = y_hat), colour = "steelblue2", size = 1) +
  labs(x = "Year", y = "Number of reported incidents")
week_plot
```

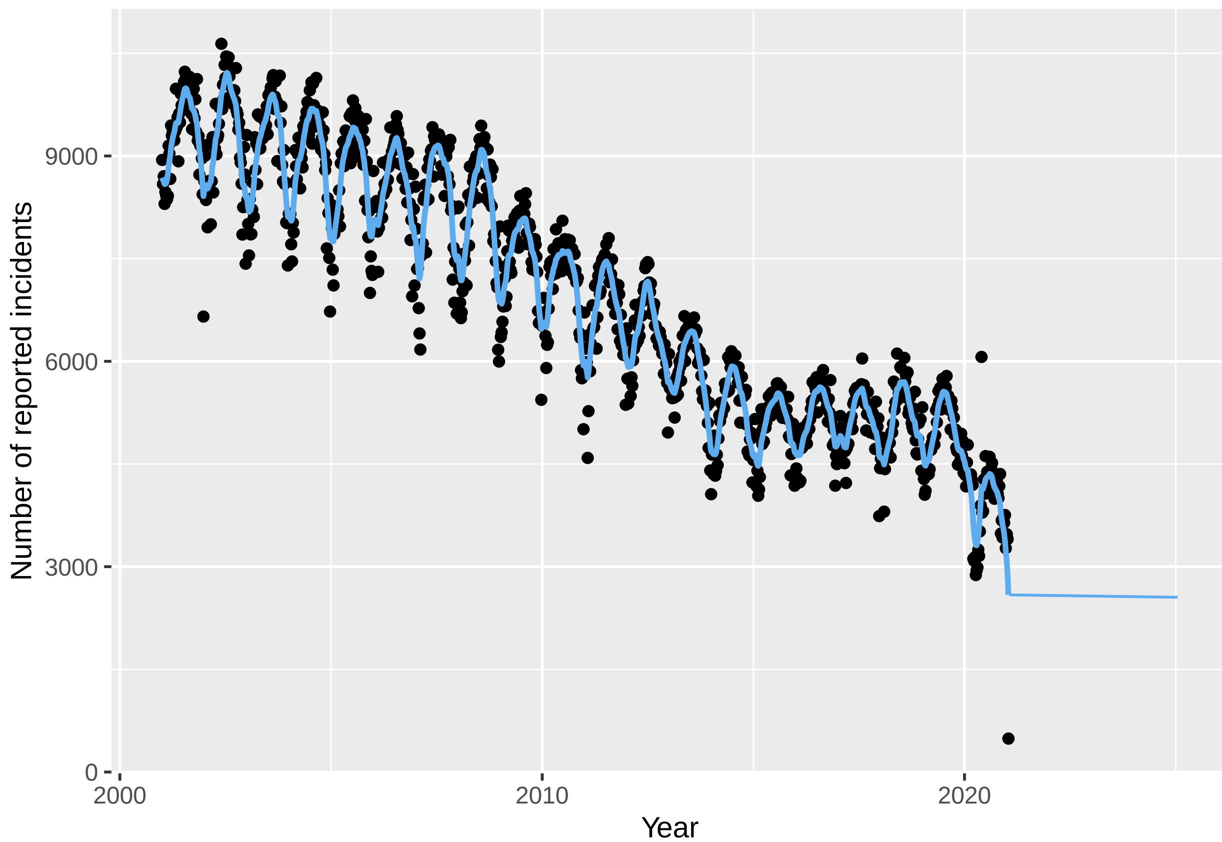


The plot suggests a cyclic pattern over years, so naturally we proceed to consider using a trigonometric transformation of the data (taking year as the period), in the hope that we can capture this periodicity with a simpler model. Note that, whilst kernel ridge regression is useful for interpolation, this model with a radial basis kernel actually has limited utility for forecasting reported crime counts into the future. We illustrate this point by attempting to extrapolate from this model.

```r
# Get the start of the next week not included in the data
start_week_date <- max(df$week_start) + 7
# Get the corresponding week number
start_week_int <- max(df$cumulative_week) + 1

extra_dates <- seq(ymd(start_week_date), ymd(start_week_date + 365*4),
                   by = '1 week')
extra_X <- start_week_int:(start_week_int + length(extra_dates) - 1)
extra_X <- (extra_X - attributes(X)$`scaled:center`) /
```

```
  attributes(X)$`scaled:scale`
# Predict using our fitted model
extra_y_hat <- kr_best$predict(as.matrix(extra_X))
# Attach these values to the data frame
extra_df <- tibble(extra_dates, extra_y_hat)
week_plot + geom_line(data = extra_df, aes(extra_dates, extra_y_hat), colour = "steelblue2")
```



The cycles are captured solely due to being able to interpolate locally between points. When extrapolating with a radial basis function kernel, no cycles are predicted, so the out-of-sample accuracy drops substantially. The cross-validation error found is only applicable if the new data followed the same distribution, and this is simply not the case when considering future data.

Monthly Predictions with a Spatial Component

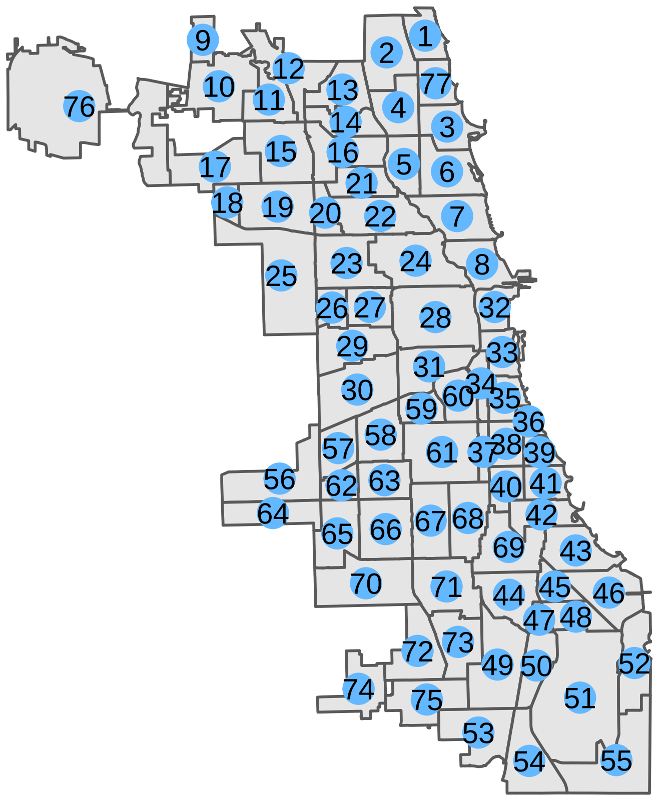
The previous example aggregated counts of reported crime over weeks, and as such used only a single-dimensional feature vector. We look at extending this analysis by aggregating over an additional spatial component. In the ensuing analysis, we consider predicting the reported crime counts over a month, aggregated over the community areas.

The positions of the community areas will be represented by the coordinates of their centroids, which are extracted from their shapefiles.

```
data("community_bounds")
centroids <- community_bounds %>%
  arrange(as.integer(area_numbe)) %>%
  st_centroid() %>%
  st_coordinates() %>%
```

```
  as_tibble() %>%
  mutate(community_area = 1:nrow(.))

centroids_plot <- ggplot() +
  geom_sf(data = community_bounds) +
  geom_point(data = centroids, aes(X, Y), size = 5, colour = "steelblue1") +
  geom_text(data = centroids, aes(X, Y, label = community_area)) +
  theme_void()
centroids_plot
```



We shall consider data from the years 2016 to 2020.

```
df <- load_data(c(2016, 2020), strings_as_factors = FALSE, na_omit = TRUE)
```

```
df %<>% select(date, community_area) %>%
  mutate(date = date(date), year = year(date), month = month(date)) %>%
  group_by(year, month) %>%
  mutate(date = min(date)) %>%
  group_by_all() %>%
  summarise(n = n()) %>%
  left_join(centroids, by = "community_area")
```

In order to choose a suitable bandwidth parameter for the radial basis function, we perform repeated 5-fold cross-validation.

```
# Obtain data matrix for fitting the model
X <- df %>% ungroup() %>%
```

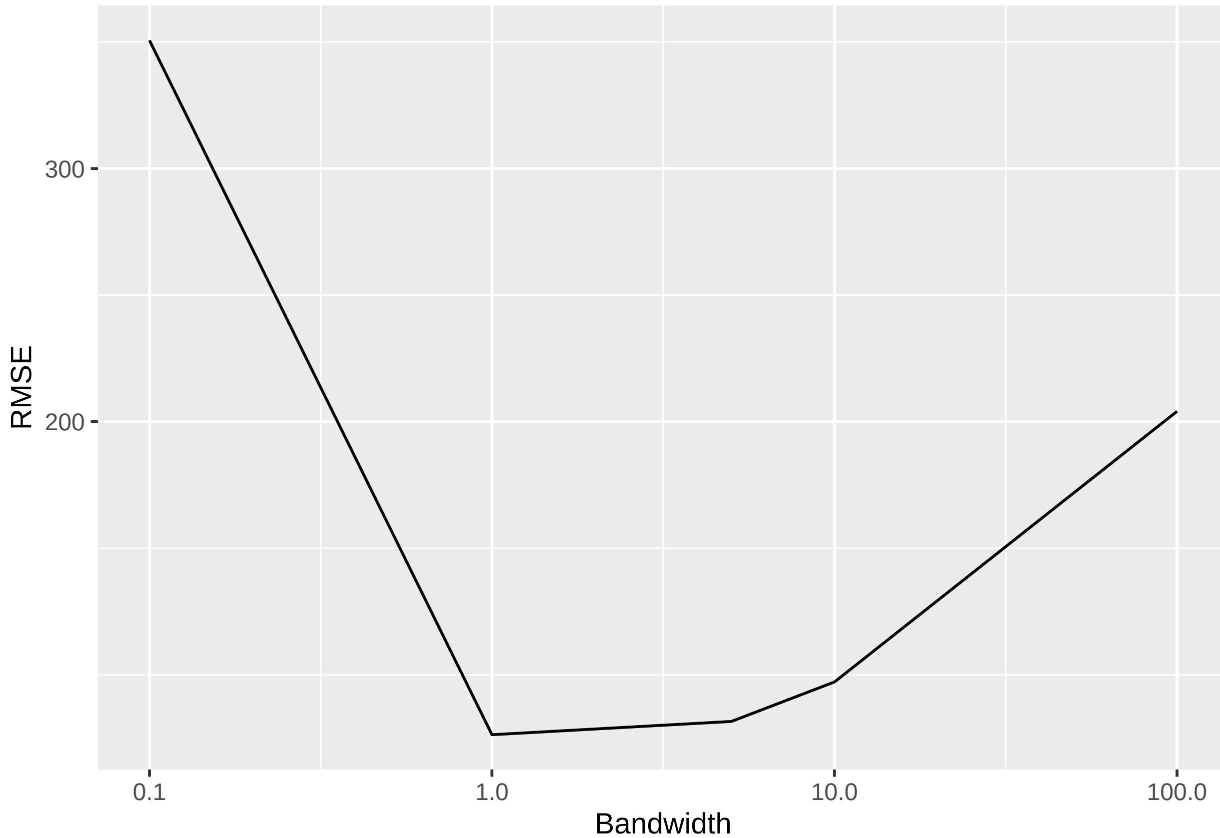
```r
  select(-community_area, -date, -n) %>%
  as.matrix()
# Scale the parameters
X <- scale(X)
y <- df$n
```

```r
# Bandwidth candidates
bandwidth <- c(0.1, 1, 5, 10, 100)

rmse_vec <- c()
# Set new seed for reproduciblility
set.seed(3)
for (i in 1:length(bandwidth)){
  kr <- KernelRidge$new("rbf", lambda = 0.001, bandwidth[i])
  cv_results <- kfold_cv(kr, X, y, metrics, k = 5, n_reps = 5,
                         parallel = TRUE, n_threads = 5)
  mean_rmse <- mean(cv_results$rmse)
  rmse_vec <- c(rmse_vec, mean_rmse)
  # Save best model
  if (mean_rmse == min(rmse_vec)){
    kr_best <- kr$clone(deep=TRUE)
    cv_results_best <- cv_results
  }
}

rbf_results <- data.frame(bandwidth, rmse = rmse_vec)

band_plot <- ggplot(rbf_results) +
  geom_line(aes(x = bandwidth, y = rmse)) +
  scale_x_log10() +
  scale_y_log10()
band_plot
```



We obtain an optimal value of 1 which minimises the mean square error. Again, we can look at the results for the three repeats of 5-fold cross-validation for this model:

```
xtable::xtable(as_tibble(cv_results_best))
```

```
## % latex table generated in R 4.0.3 by xtable 1.8-4 package
## % Sun Jan 24 16:58:41 2021
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrr}
##   \hline
##  & rmse & r2 \\
##   \hline
## 1 & 122.86 & 0.75 \\
##   2 & 120.33 & 0.76 \\
##   3 & 119.14 & 0.77 \\
##   4 & 122.10 & 0.75 \\
##   5 & 121.17 & 0.76 \\
##    \hline
## \end{tabular}
## \end{table}
```

On average, this model explained 75.73% of the variance. Note the lower R^2 value: in contrast to the weekly model, observations were not averaged over the whole of Chicago.

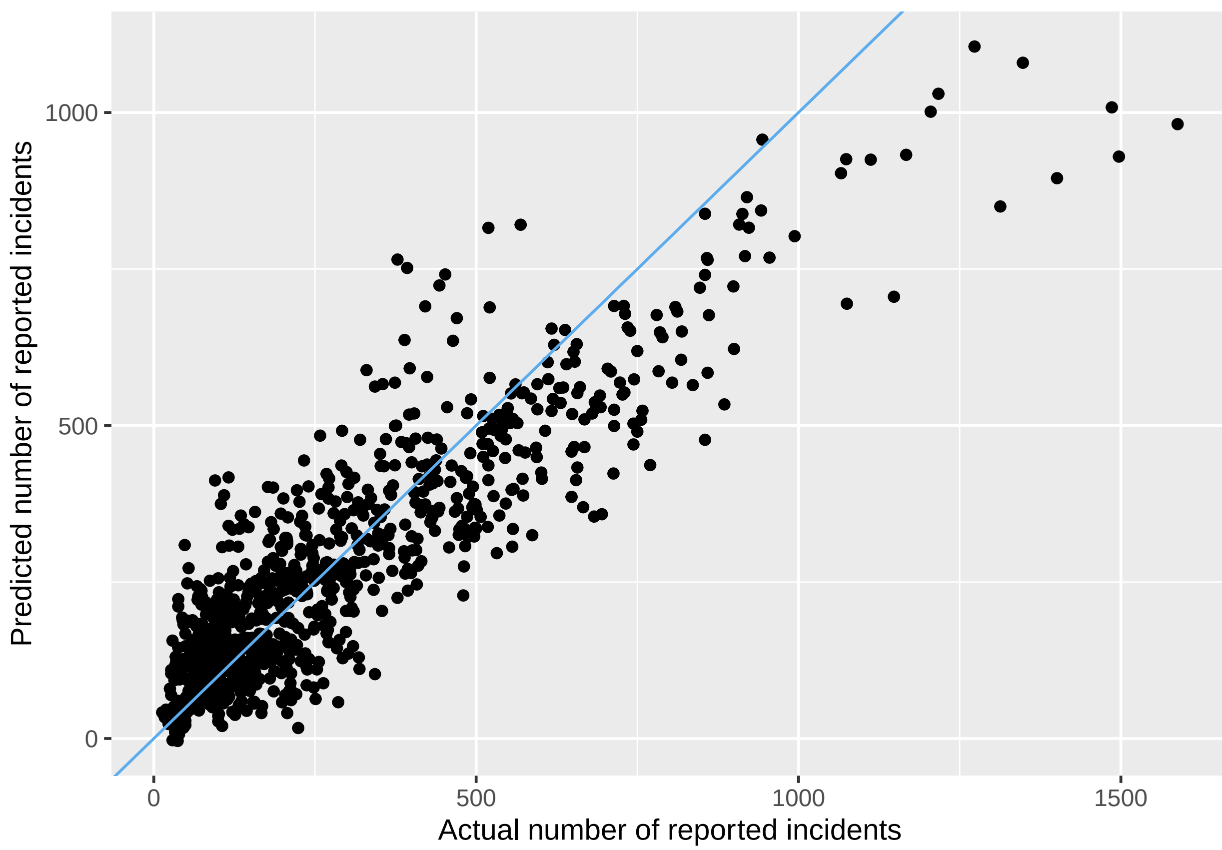
Although being higher dimensional so difficult to plot, we can plot the predicted values against the observed values on a held out data set:

```
index <- sample(1:nrow(X), round(0.2 * nrow(X)))

X_train <- X[-index, , drop = FALSE]
y_train <- y[-index]
X_test <- X[index, ]
y_test <- y[index]

# Refit best model on full training dataset
kr_best$fit(X_train, y_train)
y_hat <- kr_best$predict(X_test)

pred_plot <- qplot(y_test, y_hat, geom = "point") +
  geom_abline(slope = 1, intercept = 0, colour = "steelblue2") +
  labs(x = "Actual number of reported incidents", y = "Predicted number of reported incidents") +
  scale_y_continuous(breaks = seq(0, 2000, by = 500))
pred_plot
```



The plot illustrates strong predictive power, particularly when the number of reported incidents is smaller. We do note the drop-off at the higher end of actual reported incidents, suggesting the model tends to under-predict in these cases. This is to be expected somewhat, as extremely high counts are rare and the model has not been trained on a dataset containing such high counts.

We conclude this section by noting that kernel ridge regression can provide prediction functions that explain a high proportion of the variance. However, a major limitation of this method in our case is that it would be primarily be useful for interpolation, rather than extrapolation, for the reasons outlined above.

In the subsequent section, we explore generalised additive models, which form a more principled approach for forecasting the number of crimes.

Generalised Additive Models