# Police Attendance Model

*Brendan Nolan*

*16/04/2018*

### Setup

The task is to produce a model which will predict whether or not a police officer will attend the scene of a car accident. For this, I am using the 2014 UK government road safety data, available from gov.co.uk.

First I load the required packages and import the data, renaming the columns to make them consistent with R's naming conventions and to make them easier to work with. I also set a seed for random number generation.

```
pkgs <- c("tidyverse", "mlr", "janitor", "lubridate", "here")
invisible(lapply(pkgs, library, character.only = TRUE))
set.seed(1)
```

```
raw_data <- read_csv(here("DfTRoadSafety_Accidents_2014.csv")) %>%
  clean_names()
#> Parsed with column specification:
#> cols(
#>   .default = col_integer(),
#>   Accident_Index = col_character(),
#>   Longitude = col_double(),
#>   Latitude = col_double(),
#>   Date = col_character(),
#>   Time = col_time(format = ""),
#>   `Local_Authority_(Highway)` = col_character(),
#>   LSOA_of_Accident_Location = col_character()
#> )
#> See spec(...) for full column specifications.
glimpse(raw_data)
#> Observations: 146,322
#> Variables: 32
#> $ accident_index                               <chr> "201401BS70001", "...
#> $ location_easting_osgr                        <int> 524600, 525780, 52...
#> $ location_northing_osgr                       <int> 179020, 178290, 17...
#> $ longitude                                    <dbl> -0.206443, -0.1897...
#> $ latitude                                     <dbl> 51.49634, 51.48952...
#> $ police_force                                 <int> 1, 1, 1, 1, 1, 1, ...
#> $ accident_severity                            <int> 3, 3, 3, 3, 3, 3, ...
#> $ number_of_vehicles                           <int> 2, 2, 2, 1, 2, 3, ...
#> $ number_of_casualties                         <int> 1, 1, 1, 1, 1, 1, ...
#> $ date                                         <chr> "09/01/2014", "20/...
#> $ day_of_week                                  <int> 5, 2, 3, 4, 5, 6, ...
#> $ time                                         <time> 13:21:00, 23:00:0...
#> $ local_authority_district                     <int> 12, 12, 12, 12, 12...
#> $ local_authority_highway                      <chr> "E09000020", "E090...
#> $ x1st_road_class                              <int> 3, 3, 3, 5, 3, 3, ...
#> $ x1st_road_number                             <int> 315, 3218, 308, 0,...
#> $ road_type                                    <int> 6, 6, 6, 6, 6, 2, ...
#> $ speed_limit                                  <int> 30, 30, 30, 30, 30...
```

```
#> $ junction_detail                             <int> 0, 5, 3, 3, 7, 0, ...
#> $ junction_control                            <int> -1, 4, 4, 4, 4, -1...
#> $ x2nd_road_class                             <int> -1, 3, 6, 6, 3, -1...
#> $ x2nd_road_number                            <int> 0, 3220, 0, 0, 4, ...
#> $ pedestrian_crossing_human_control           <int> 0, 0, 0, 0, 0, 0, ...
#> $ pedestrian_crossing_physical_facilities     <int> 0, 5, 0, 1, 8, 0, ...
#> $ light_conditions                            <int> 1, 7, 1, 4, 1, 1, ...
#> $ weather_conditions                          <int> 2, 1, 1, 1, 1, 1, ...
#> $ road_surface_conditions                     <int> 2, 1, 1, 1, 1, 1, ...
#> $ special_conditions_at_site                  <int> 0, 0, 0, 0, 0, 0, ...
#> $ carriageway_hazards                         <int> 0, 0, 0, 0, 0, 0, ...
#> $ urban_or_rural_area                         <int> 1, 1, 1, 1, 1, 1, ...
#> $ did_police_officer_attend_scene_of_accident <int> 2, 2, 1, 2, 1, 1, ...
#> $ lsoa_of_accident_location                   <chr> "E01002814", "E010...
```

Three challenges are evident immediately:

- There is a very large number of observations.
- There is a very large number of variables.
- Most variables are categorical, which will necessitate many dummy variables if I choose a logistic-regression-type technique.

## Data Preparation

Next, I make sure that the data which should be numerical is in numeric format and that the categorical data is in factor format. In this step, I also remove a few columns:

- `accident_index` is just an identifier, so there's no meaningful information there
- `location_easting_osgr` and `location_northing_osgr` are just different measures of `latitude` and `longitude` so I remove them (and keep `latitude` and `longitude`)
- `x1st_road_number`, `x2nd_road_class` and `x2nd_road_number`, `local_authority_highway` and `lsoa_of_accident_location` are categorical variables with too many levels (relative to the number of samples in the data) for each level to have a significant meaning, so I remove them.
- `local_authority_district` captures regional information which is highly correlated with that of `police_force`, so I remove `local_authority_district` and keep `police_force`.

I do not code `time` as numeric because this misses the fact that 23:59 is very close to 00:00. Instead I use `hour_of_day` (categorical variable with 24 levels). For similar reasons, I convert `date` to `month`.

```r
transformed_data <- raw_data %>%
  transmute(
    longitude = longitude,
    latitude = latitude,
    police_force = as.factor(police_force),
    accident_severity = as.factor(accident_severity),
    number_of_vehicles = number_of_vehicles,
    number_of_casualties = number_of_casualties,
    month = as.factor(month(dmy(date))),
    day_of_week = as.factor(day_of_week),
    hour_of_day = as.factor(hour(hms(time))),
    first_road_class = as.factor(x1st_road_class),
    road_type = as.factor(road_type),
    speed_limit = as.factor(speed_limit),
    junction_detail = as.factor(junction_detail),
    junction_control = as.factor(junction_control),
```

```r
    pedestrian_crossing_human_control =
      as.factor(pedestrian_crossing_human_control),
    pedestrian_crossing_physical_facilities =
      as.factor(pedestrian_crossing_physical_facilities),
    light_conditions = as.factor(light_conditions),
    weather_conditions = as.factor(weather_conditions),
    road_surface_conditions = as.factor(road_surface_conditions),
    special_conditions_at_site = as.factor(special_conditions_at_site),
    carriageway_hazards = as.factor(carriageway_hazards),
    urban_or_rural_area = as.factor(urban_or_rural_area),
    did_police_officer_attend_scene_of_accident =
      as.factor(did_police_officer_attend_scene_of_accident)
    ) %>%
  as.data.frame() %>%  # mlr prefers data frames to tibbles
  removeConstantFeatures() %>%
  drop_na()  # remove rows with missing values
```

Next, I (randomly) create the indices of the training and test sets.

```r
nr <- nrow(transformed_data)
train_indices <- sample.int(nr, nr * (2 / 3))
test_indices <- setdiff(seq_len(nr), train_indices)
```

### The simplest possible model

The simplest possible model would say that a police officer always attends the scene of a traffic accident. I will crudely compare other models to this, in order to get an impression of whether or not those models are effective. I would like to estimate the test error rate of this model. Since there is no fitting (hence no danger of overfitting) I can estimate the test error rate by the training error rate (note that `did_police_officer_attend_scene_of_accident` is coded as `1` for `TRUE` and `2` for FALSE.):

```r
mean(transformed_data[train_indices, ]$did_police_officer_attend_scene_of_accident == 2)
#> [1] 0.1825665
```

### Penalized logistic regression

The first thing I try is a penalized logistic regression. Penalizing is designed to avoid over-fitting, which is a big danger with so many variables. LASSO penalised logistic regression also reduces the number of variables by setting some coefficients to zero.

I use the `mlr` (*Machine Learning with R*) package for this.

First I set up the generic machine learning task at hand, with the data and target variable. I also set up a sub-task for tuning the hyperparameter (choosing the best hyperparameter `lambda1` for the lasso-penalized logistic regression).

```r
generic_task <- makeClassifTask("road_safety", data = transformed_data,
                    target = "did_police_officer_attend_scene_of_accident") %>%
  removeConstantFeatures()

tune_task <- generic_task %>%
  subsetTask(train_indices) %>%
  removeConstantFeatures()
```

Next I set up the LASSO penalized logistic regression learning procedure, including the possible choices of hyperparameter `lambda1` that I want to iterate over in order to choose the best one via 5-fold cross-validation. For the sake of time, I choose a small set of possible values for `lambda1`, using powers of 2 in order to have values spread over a relatively wide range. I create the learner with `makePreprocWrapperCaret()`. Creating a learner with `makePreprocWrapperCaret()` has the following advatnage: when called, such a learner will automatically preprocess the data in the appropriate fashion.

At this point, it is worth acknowledging that I may be missing the most effective values of `lambda1`. I choose high values of `lambda1` because for low values - particularly those below 1 - the code takes a very long time to run. The long runtime is likely at least partially due to the large number of variables and in particular to the large number of categorical variables, many of which have a large number of levels; as such, the regression requires the coding of an extremely large number of dummy variables. This necessitates the fitting of almost two-hundred coefficients, which is inevitably slow and possibly quite unreliable.

```
learner_plr <- makePreprocWrapperCaret("classif.penalized")
#> Loading required package: penalized
#> Loading required package: survival
#> Welcome to penalized. For extended examples, see vignette("penalized").
resamp <- makeResampleDesc("CV", iters = 5,
                           stratify = TRUE)
lambda1_set <- makeParamSet(makeDiscreteParam("lambda1",
                                              values = 2 ^ (7:5)))
ctrl <- makeTuneControlGrid()
```

Next I do the parameter tuning:

```
tuned_plr <- tuneParams(learner_plr, tune_task, resampling = resamp,
                        par.set = lambda1_set, control = ctrl)
#> [Tune] Started tuning learner classif.penalized.preproc for parameter set:
#>             Type len Def    Constr Req Tunable Trafo
#> lambda1 discrete   -   - 128,64,32   -    TRUE     -
#> With control class: TuneControlGrid
#> Imputation value: 1
#> [Tune-x] 1: lambda1=128
#> [Tune-y] 1: mmce.test.mean=0.1793681; time: 0.5 min
#> [Tune-x] 2: lambda1=64
#> [Tune-y] 2: mmce.test.mean=0.1795116; time: 1.8 min
#> [Tune-x] 3: lambda1=32
#> [Tune-y] 3: mmce.test.mean=0.1791631; time: 5.7 min
#> [Tune] Result: lambda1=32 : mmce.test.mean=0.1791631
```

The best `lambda1` is given by `tuned_plr$x$lambda1` - namely 32 - so I will go with that. The `mmce` (*Mean MisClassification Error*) was 0.1791631. So now I can choose the best penalized logistic regression model (subject to the range of possible hyperparameters which I allowed):

```
best_learner_plr <- setHyperPars(learner_plr, par.vals = tuned_plr$x)
model_plr <- train(best_learner_plr, generic_task, subset = train_indices)
```

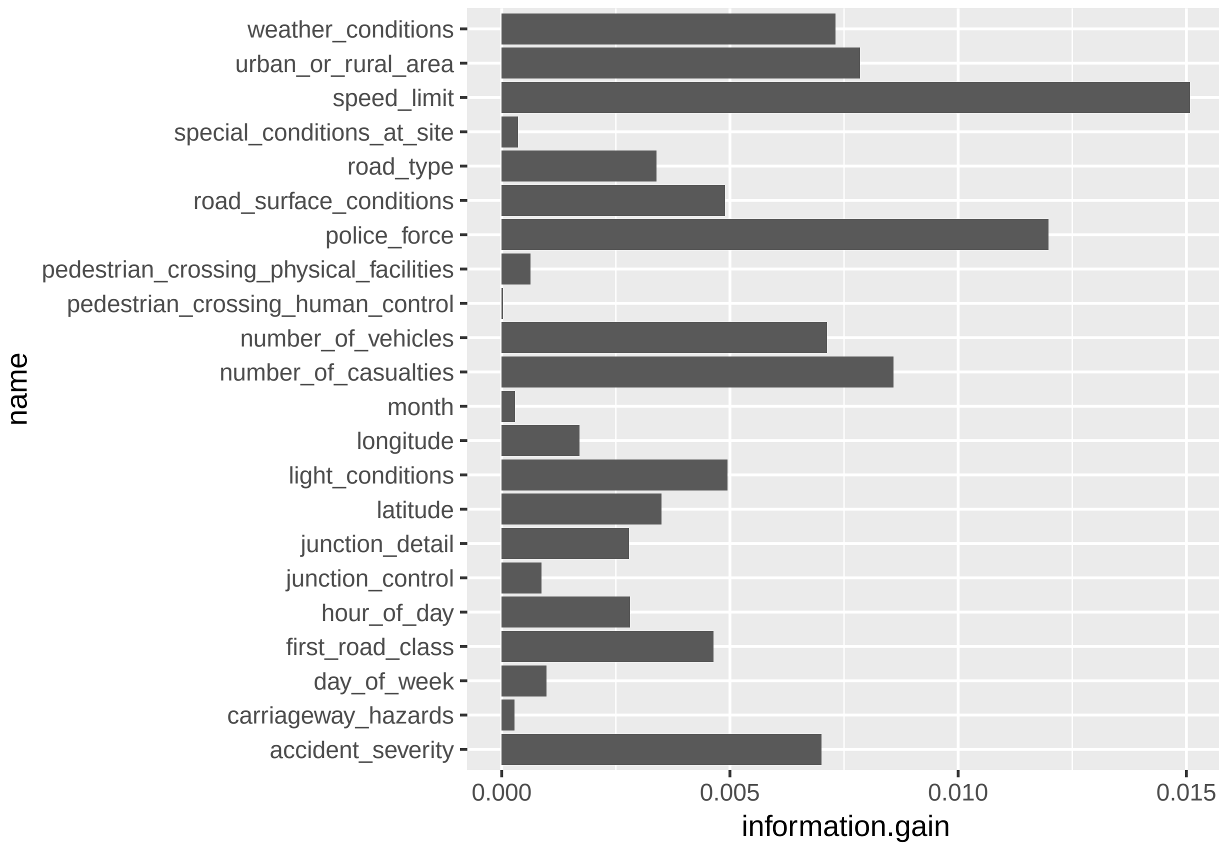
As I said above, the mmce of this `model_plr` is 0.1791631, which is the estimated test error rate. This does not improve much upon the simplest possible model (which assumes that a police officer always turns up), whose estimated test error rate was computed above to be 0.1825665. Nevertheless, `model_plr` is slightly better and I may be able to use it for inference. Hopefully, the LASSO model will have set most of the almost-two-hundred fitted coefficients to zero:

```
getLearnerModel(model_plr, more.unwrap = TRUE)
#> Penalized logistic regression object
#> 180 regression coefficients of which 95 are non-zero
```

```
#>
#> Loglikelihood =   -41099.84
#> L1 penalty =  718.5474   at lambda1 =  32
```

That still leaves me with a very large number of nonzero regression coefficients and it is difficult to do much inference here. The best I can do is to graph the information gain of each of the variables.

```
generateFilterValuesData(tune_task, method = "information.gain")$data %>%
  ggplot(aes(x = name, y = information.gain)) +
  geom_bar(stat = "identity") +
  coord_flip()
```




Given the limitations of my penalized logistic regression approach and the fact that it is not much better than the model that predicts a police officer always attends, I will now try gradient boosting as an alternative method.

## Gradient Boosted Method

In order to choose the best boosted trees method, I will again choose a set of hyperparameters to iterate over in order to choose the best ones via 5-fold cross validation. If I had more time, I would choose greater possible ranges of the hyperparameters.

```
learner_gbm <- makePreprocWrapperCaret("classif.gbm")
param_set_gbm <- makeParamSet(
                  makeDiscreteParam("n.trees", values = 50 * 2 ^ (0:4)),
                  makeDiscreteParam("interaction.depth", values = 1:2),
```

```r
                   makeDiscreteParam("shrinkage", values = 2 ^ (-3:-4)))
tuned_gbm <- tuneParams(learner_gbm, task = tune_task, resampling = resamp,
                        par.set = param_set_gbm, control = ctrl)
#> [Tune] Started tuning learner classif.gbm.preproc for parameter set:
#>                        Type len Def             Constr Req Tunable Trafo
#> n.trees            discrete   -   - 50,100,200,400,800   -    TRUE     -
#> interaction.depth discrete   -   -                1,2   -    TRUE     -
#> shrinkage         discrete   -   -       0.125,0.0625   -    TRUE     -
#> With control class: TuneControlGrid
#> Imputation value: 1
#> [Tune-x] 1: n.trees=50; interaction.depth=1; shrinkage=0.125
#> [Tune-y] 1: mmce.test.mean=0.1795424; time: 0.4 min
#> [Tune-x] 2: n.trees=100; interaction.depth=1; shrinkage=0.125
#> [Tune-y] 2: mmce.test.mean=0.1791426; time: 0.6 min
#> [Tune-x] 3: n.trees=200; interaction.depth=1; shrinkage=0.125
#> [Tune-y] 3: mmce.test.mean=0.1787120; time: 1.0 min
#> [Tune-x] 4: n.trees=400; interaction.depth=1; shrinkage=0.125
#> [Tune-y] 4: mmce.test.mean=0.1784557; time: 1.8 min
#> [Tune-x] 5: n.trees=800; interaction.depth=1; shrinkage=0.125
#> [Tune-y] 5: mmce.test.mean=0.1784250; time: 3.4 min
#> [Tune-x] 6: n.trees=50; interaction.depth=2; shrinkage=0.125
#> [Tune-y] 6: mmce.test.mean=0.1784455; time: 0.4 min
#> [Tune-x] 7: n.trees=100; interaction.depth=2; shrinkage=0.125
#> [Tune-y] 7: mmce.test.mean=0.1777894; time: 0.7 min
#> [Tune-x] 8: n.trees=200; interaction.depth=2; shrinkage=0.125
#> [Tune-y] 8: mmce.test.mean=0.1777894; time: 1.4 min
#> [Tune-x] 9: n.trees=400; interaction.depth=2; shrinkage=0.125
#> [Tune-y] 9: mmce.test.mean=0.1776664; time: 2.5 min
#> [Tune-x] 10: n.trees=800; interaction.depth=2; shrinkage=0.125
#> [Tune-y] 10: mmce.test.mean=0.1779739; time: 5.2 min
#> [Tune-x] 11: n.trees=50; interaction.depth=1; shrinkage=0.0625
#> [Tune-y] 11: mmce.test.mean=0.1825665; time: 0.3 min
#> [Tune-x] 12: n.trees=100; interaction.depth=1; shrinkage=0.0625
#> [Tune-y] 12: mmce.test.mean=0.1795526; time: 0.5 min
#> [Tune-x] 13: n.trees=200; interaction.depth=1; shrinkage=0.0625
#> [Tune-y] 13: mmce.test.mean=0.1790401; time: 0.9 min
#> [Tune-x] 14: n.trees=400; interaction.depth=1; shrinkage=0.0625
#> [Tune-y] 14: mmce.test.mean=0.1787633; time: 1.8 min
#> [Tune-x] 15: n.trees=800; interaction.depth=1; shrinkage=0.0625
#> [Tune-y] 15: mmce.test.mean=0.1785275; time: 3.4 min
#> [Tune-x] 16: n.trees=50; interaction.depth=2; shrinkage=0.0625
#> [Tune-y] 16: mmce.test.mean=0.1792246; time: 0.4 min
#> [Tune-x] 17: n.trees=100; interaction.depth=2; shrinkage=0.0625
#> [Tune-y] 17: mmce.test.mean=0.1784660; time: 0.7 min
#> [Tune-x] 18: n.trees=200; interaction.depth=2; shrinkage=0.0625
#> [Tune-y] 18: mmce.test.mean=0.1775946; time: 1.3 min
#> [Tune-x] 19: n.trees=400; interaction.depth=2; shrinkage=0.0625
#> [Tune-y] 19: mmce.test.mean=0.1775536; time: 2.5 min
#> [Tune-x] 20: n.trees=800; interaction.depth=2; shrinkage=0.0625
#> [Tune-y] 20: mmce.test.mean=0.1774921; time: 4.6 min
#> [Tune] Result: n.trees=800; interaction.depth=2; shrinkage=0.0625 : mmce.test.mean=0.1774921
```

I see that the best `n.trees`, `interaction.depth`, and `shrinkage` were respectively 800, 2, and 0.0625, so I'll go with those. The `mmce` was 0.1774921. So now I can choose the best GBM model (subject to my hyperparameter choice):

```
best_learner_gbm <- setHyperPars(learner_gbm, par.vals = tuned_gbm$x)
model_gbm <- train(best_learner_gbm, generic_task, subset = train_indices)
#> Distribution not specified, assuming bernoulli ...
```

This is slightly better (in terms of mmce) than my chosen penalized logistic regression model, so I will choose `model_gbm` as my final model. I now check how it performs on the test set:

```
task_pred_gbm <- predict(model_gbm, task = generic_task, subset = test_indices)
glimpse(as.data.frame(task_pred_gbm))
#> Observations: 48,774
#> Variables: 3
#> $ id       <int> 3, 4, 5, 9, 10, 16, 18, 20, 23, 25, 31, 32, 34, 35, 4...
#> $ truth    <fct> 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,...
#> $ response <fct> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
pred_gbm_df <- as.data.frame(task_pred_gbm) %>%
  mutate(truth = if_else(truth == "1", "Y", "N"),
         response = if_else(response == "1", "Y", "N"))
head(pred_gbm_df)
#>   id truth response
#> 1  3     Y        Y
#> 2  4     N        Y
#> 3  5     Y        Y
#> 4  9     Y        Y
#> 5 10     Y        Y
#> 6 16     Y        Y
group_by(pred_gbm_df, response) %>% summarise(n = n())
#> # A tibble: 2 x 2
#>   response     n
#>   <chr>    <int>
#> 1 N          917
#> 2 Y        47857
conf_mat <- calculateConfusionMatrix(task_pred_gbm)
conf_mat
#>         predicted
#> true         1   2 -err.-
#>   1      39567 301    301
#>   2       8290 616   8290
#>   -err.-  8290 301   8591
```

(When interpreting this confusion matrix, recall that `did_police_officer_attend_scene_of_accident` is coded as `1` for `TRUE` and `2` for FALSE.)

## Conclusion

I can conclude quite a lot from the confusion matrix `conf_mat`: if I view a positive as a police officer attending an accident, then the model `model_gbm` is very sensitive - almost every time a police officer attended, this was predicted by the model. However the model is not very specific - almost every time a police officer did not attend, the model failed to predict this. So the model has a poor sensitivity-specificity tradeoff. In spite of its drawbacks, the model is redeemed somewhat by its negative predictive value. It is quite a bold claim to predict that a police officer will not attend the scene of an accident and the model is quite accurate when it makes this claim: it has a negative predictive value of 0.6717557. (Negative predictive value is defined as the proportion of negative predictions which are correct.)

Here is what I think is happening: even though many combinations of the variables can reduce the probability that a police officer will attend the scene of an accident, this probability is almost never below 0.5, so that it is almost always wise to predict that a police officer will attend. Indeed my chosen model `model_gbm` almost always predicts that a police officer will attend.