07_2_Machine_Learning_Features

Yue Xiong, LMU, yue.xiong@stat.uni-muenchen.de

Machine Learning – Creating Features

Introduction

In this tutorial, we'll discuss how to formulate a policy problem or a social science question in the machine learning framework; how to transform raw data into something that can be fed into a model; how to build, evaluate, compare, and select models; and how to reasonably and accurately interpret model results.

This tutorial is based on chapter "Machine Learning" of Big Data and Social Science.

Setup

```
library('dbplyr')  # load the corresponding libraries
library('dplyr')
library('RSQLite')
library('glue')
```

```
# Establish a connection to the ncdoc.db database
database_path = "F:/hiwi_work_notebook/ncdoc.db"
conn = DBI::dbConnect(SQLite(), database_path)
```

```
# Checking the tables saved in this database
src_dbi(conn)
```

```
## src:  sqlite 3.36.0 [F:\hiwi_work_notebook\ncdoc.db]
## tbls: feature_age_2008, feature_age_first_admit, feature_agefirstadmit,
##   feature_length_sentence_2000_2008, feature_num_admits_2000_2008,
##   features_2000_2008, inmate, offender, recidivism_labels_2009_2013,
##   recidivism_labels_2014_2018, sentences, sentences_prep
```

Problem Formulation

Our Machine Learning Problem >Of all prisoners released, we would like to predict who is likely to reenter jail within *5* years of the day we make our prediction. For instance, say it is Jan 1, 2009 and we want to identify which >prisoners are likely to re-enter jail between now and end of 2013. We can run our predictive model and identify who is most likely at risk. The is an example of a *binary classification* problem.

Note the outcome window of 5 years is completely arbitrary. You could use a window of 5, 3, 1 years or 1 day.

In order to predict recidivism, we will be using data from the `inmate` and `sentences` table to create labels (predictors, or independent variables, or X variables) and features (dependent variables, or Y variables).

We need to munge our data into **labels** (1_Machine_Learning_Labels.rmd) and **features** (2_Machine_Learning_Features.rmd) before we can train and evaluate **machine learning models** (3_Machine_Learning_Models.rmd).

This notebook assumes that you have already worked through the `1_Machine_Learning_Labels` R markdown notebook. If that is not the case, you can execute the according notebook as specified.

Feature Generation

Our features for prediction recidivism (between 2009 and 2013) are the following:

- `num_admits`: The number of times someone has been admitted to prison before 2009. The more times someone has been to prison the more times they are likely continue to be arrested.
- `length_longest_sentence`: The length of the longest sentence of all admits before 2009. Long previous sentences might decrease the likelihood of future arrests.
- `age_first_admit`: The age someone was first admitted to prison. The idea behind creating this feature is that people who are younger when they are first arrested are more likely to be arrested again.
- `age`: The age at the end of our last exit time range, i.e. in 2008. People who are younger when they are released might be more likely to be arrested again.

First, we create a new sentence table `sentences_prep` that includes the sentence begin and end dates in date format.

```
sql_string = "drop table if exists sentences_prep;"
DBI::dbSendStatement(conn, sql_string)
```

```
## <SQLiteResult>
##   SQL  drop table if exists sentences_prep;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string ="
create table sentences_prep as
select inmate_doc_number,
cast(inmate_sentence_component as integer) as sentence_component,
date([sentence_begin_date_(for_max)]) as sentence_begin_date,
date(actual_sentence_end_date) as sentence_end_date
from sentences;
"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create table sentences_prep as
## select inmate_doc_number,
```

```
## cast(inmate_sentence_component as integer) as sentence_component,
## date([sentence_begin_date_(for_max)]) as sentence_begin_date,
## date(actual_sentence_end_date) as sentence_end_date
## from sentences;
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

To create the feature `num_admits`, we count the number of rows (individual sentence periods) for each `inmate_doc_number` before 2009 and write this information into `feature_num_admits_2000_2008`.

```
sql_string = "drop table if exists feature_num_admits_2000_2008;"
DBI::dbSendStatement(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  drop table if exists feature_num_admits_2000_2008;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string ="
create table feature_num_admits_2000_2008 as
select inmate_doc_number, count(*) num_admits
from sentences_prep
where inmate_doc_number in (select inmate_doc_number from recidivism_labels_2009_2013)
and sentence_begin_date < '2008-12-31' and sentence_component = 1
group by inmate_doc_number;
"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create table feature_num_admits_2000_2008 as
## select inmate_doc_number, count(*) num_admits
## from sentences_prep
## where inmate_doc_number in (select inmate_doc_number from recidivism_labels_2009_2013)
## and sentence_begin_date < '2008-12-31' and sentence_component = 1
## group by inmate_doc_number;
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

For `length_longest_sentence`, we first compute the length of all sentences before 2009 and create the table `feature_length_sentence_2000_2008`.

```
sql_string = "drop table if exists feature_length_sentence_2000_2008;"
DBI::dbSendStatement(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  drop table if exists feature_length_sentence_2000_2008;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string ="
create table feature_length_sentence_2000_2008 as
select inmate_doc_number, sentence_component, cast(julianday(sentence_end_date) - julianday(sentence_beg
from sentences_prep
where inmate_doc_number in (select inmate_doc_number from recidivism_labels_2009_2013)
and sentence_begin_date < '2008-12-31' and sentence_component = 1
and sentence_begin_date > '0001-01-01' and sentence_end_date > '0001-01-01' and sentence_end_date > sent
"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create table feature_length_sentence_2000_2008 as
## select inmate_doc_number, sentence_component, cast(julianday(sentence_end_date) - julianday(sentence_
## from sentences_prep
## where inmate_doc_number in (select inmate_doc_number from recidivism_labels_2009_2013)
## and sentence_begin_date < '2008-12-31' and sentence_component = 1
## and sentence_begin_date > '0001-01-01' and sentence_end_date > '0001-01-01' and sentence_end_date > s
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

On this basis, we find the longest sentence period (`max(length_sentence)`) for each `inmate_doc_number`.

```
sql_string = "drop table if exists feature_length_long_sentence_2000_2008;"
DBI::dbSendStatement(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  drop table if exists feature_length_long_sentence_2000_2008;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string ="
create temp table feature_length_long_sentence_2000_2008 as
select inmate_doc_number, max(length_sentence) length_longest_sentence
from feature_length_sentence_2000_2008
group by inmate_doc_number;
"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create temp table feature_length_long_sentence_2000_2008 as
## select inmate_doc_number, max(length_sentence) length_longest_sentence
## from feature_length_sentence_2000_2008
## group by inmate_doc_number;
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

There are several steps needed to compute the age at first arrest. First, we find the first arrest (`min(sentence_begin_date)`) for each `inmate_doc_number` and create the table `docnbr_admityr`.

```
sql_string = "drop table if exists docnbr_admityr;"
DBI::dbSendStatement(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  drop table if exists docnbr_admityr;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string ="
create temp table docnbr_admityr as
select inmate_doc_number, min(sentence_begin_date) min_admityr
from sentences_prep
where sentence_begin_date > '0001-01-01'
group by inmate_doc_number;
"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create temp table docnbr_admityr as
## select inmate_doc_number, min(sentence_begin_date) min_admityr
## from sentences_prep
## where sentence_begin_date > '0001-01-01'
## group by inmate_doc_number;
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

We then join the `inmate` and `docnbr_admityr` tables and extract the years from `inmate_birth_date` (birth year) and `min_admityr` (year first admitted into prison).

```
sql_string = "drop table if exists age_first_admit_birth_year;"
DBI::dbSendStatement(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  drop table if exists age_first_admit_birth_year;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
# sql_string =paste0("
# create temp table age_first_admit_birth_year as
# select da.inmate_doc_number,
# cast(strftime("%Y", da.min_admityr) as integer) min_admityr,
# cast(strftime("%Y", p.inmate_birth_date) as integer) inmate_birth_date
# from docnbr_admityr da
# left join inmate p on da.inmate_doc_number = p.inmate_doc_number;
# ")
# DBI::dbSendQuery(conn, sql_string)
```

```
create temp table age_first_admit_birth_year as
select da.inmate_doc_number,
cast(strftime("%Y", da.min_admityr) as integer) min_admityr,
cast(strftime("%Y", p.inmate_birth_date) as integer) inmate_birth_date
from docnbr_admityr da
left join inmate p on da.inmate_doc_number = p.inmate_doc_number;
```

The combined table allows us to create `age_first_admit` by subtracting the birth year from the year first admitted into prison.

```
sql_string = "drop table if exists feature_age_first_admit;"
DBI::dbSendStatement(conn, sql_string)
```

```
## <SQLiteResult>
##   SQL  drop table if exists feature_age_first_admit;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string = "
create table feature_age_first_admit as
select inmate_doc_number, (min_admityr - inmate_birth_date) age_first_admit
from age_first_admit_birth_year;"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create table feature_age_first_admit as
## select inmate_doc_number, (min_admityr - inmate_birth_date) age_first_admit
## from age_first_admit_birth_year;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

We then filter the `feature_age_first_admit` table such that it only includes observations that are observed in the label table `recidivism_labels_2009_2013`.

```
sql_string = "drop table if exists feature_agefirstadmit;"
DBI::dbSendStatement(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  drop table if exists feature_agefirstadmit;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string = "
create table feature_agefirstadmit as
select inmate_doc_number, age_first_admit
from feature_age_first_admit
where inmate_doc_number in (select inmate_doc_number from recidivism_labels_2009_2013);"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL
## create table feature_agefirstadmit as
## select inmate_doc_number, age_first_admit
## from feature_age_first_admit
## where inmate_doc_number in (select inmate_doc_number from recidivism_labels_2009_2013);
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

To compute the age in 2008, we simply subtract the `inmate_birth_date` from 2008 and store this information in `feature_age_2008`.

```
sql_string = "drop table if exists feature_age_2008;"
DBI::dbSendStatement(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  drop table if exists feature_age_2008;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
# sql_string = "
# create table feature_age_2008 as
# select inmate_doc_number, (2008 - cast(strftime("%Y", inmate_birth_date) as integer)) age
# from inmate
# where inmate_doc_number in (select inmate_doc_number from recidivism_labels_2009_2013);"
# DBI::dbSendQuery(conn, sql_string)
```

```
create table feature_age_2008 as
select inmate_doc_number, (2008 - cast(strftime("%Y", inmate_birth_date) as integer)) age
from inmate
where inmate_doc_number in (select inmate_doc_number from recidivism_labels_2009_2013);
```

Finally, we join all (final) feature tables by `inmate_doc_number` and create table `features_2000_2008`.

```
sql_string = "drop table if exists features_2000_2008;"
DBI::dbSendStatement(conn, sql_string)
```

```
## <SQLiteResult>
##   SQL  drop table if exists features_2000_2008;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
sql_string = "create table features_2000_2008 as
select f1.inmate_doc_number, f1.num_admits, f2.length_longest_sentence, f3.age_first_admit, f4.age
from feature_num_admits_2000_2008 f1
left join feature_length_long_sentence_2000_2008 f2 on f1.inmate_doc_number = f2.inmate_doc_number
left join feature_agefirstadmit f3 on f1.inmate_doc_number = f3.inmate_doc_number
left join feature_age_2008 f4 on f1.inmate_doc_number = f4.inmate_doc_number;"
DBI::dbSendQuery(conn, sql_string)
```

```
## Warning: Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  create table features_2000_2008 as
## select f1.inmate_doc_number, f1.num_admits, f2.length_longest_sentence, f3.age_first_admit, f4.age
## from feature_num_admits_2000_2008 f1
## left join feature_length_long_sentence_2000_2008 f2 on f1.inmate_doc_number = f2.inmate_doc_number
## left join feature_agefirstadmit f3 on f1.inmate_doc_number = f3.inmate_doc_number
## left join feature_age_2008 f4 on f1.inmate_doc_number = f4.inmate_doc_number;
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

We can now load the feature table and compute descriptive statistics for the features we created.

```
sql_string = "SELECT * FROM features_2000_2008"
# if tbl() used, no ; are alowed
features_2000_2008 = data.frame(tbl(conn, sql(sql_string)))
```

```
## Warning: Closing open result set, pending rows
```

```
head(features_2000_2008, n = 10)
```

```
##    INMATE_DOC_NUMBER num_admits length_longest_sentence age_first_admit age
## 1            0000028          3                    1097              19  36
## 2            0000062          2                     273              36  51
## 3            0000114          2                     256              49  56
## 4            0000133          1                    6209              23  39
```

```
## 5          0000152          1              76          37  43
## 6          0000171          2            1147          26  48
## 7          0000192          4            4282          22  56
## 8          0000193          2            1631          21  37
## 9          0000203          1            4936          53  67
## 10         0000204          7            1835          19  41
```

As with the label table, we need to create a second feature table which we will use for model evaluation purposes. We again create a function, this time called `create_features`, to ease the process. This function allows to run all feature generation steps for a given feature end date, prediction start date and prediction end date with just one function call.