Predictive Modelling Metadata Interchange Format

Release 0.1

Nicholas J. Radcliffe

Jan 07, 2022

Contents

Contents:

CHAPTER 1

Overview of the Predictive Modelling Metadata Interchange Format

The Predictive Modelling Metadata Interchange Format (PMMIF) is designed to facilitate the exchange of data for use in predictive modelling.

There are two initial use cases it seeks to support:

- datasets for conventional predictive modelling of integer, continuous and categorical outcomes
- datasets for uplift modelling of integer and continuous outcomes

PMMIF has the following goals:

- Capability to describe datasets fully and accurately
- Machine-readable metadata
- Initial expression in JSON, with future ability to support other representations such as XML, YAML, HTML etc.
- Reasonably readable in raw form by humans
- Enough extra metadata to dectect faulty reading of the data
- Ability to write programs to verify both well-formedness of the PMMIF metadata description and to verify that the data conforms to the specification contained in the metadata
- Extensibility
- Simple minimum requirements
- Inclusion of key structural information, e.g. the role that various fields play in modelling.
- Ability to describe all (or nearly all) of the datasets in the Machine Learning Repository at the University of California, Irvine, and also some example datasets for use with Uplift Modelling.
- Each `.pmm` file currently describes a single dataset, stored as a flat file, consisting of a number of records, each with a fixed set of named fields. Later enhancements will allow multiple flat files to be documented together.

1.1 Very Simple Example

Here is a very simple example of a `.pmm` file in the PMMIF format, `hillstrom3.pmm`:


```
{
    "pmmversion": "0.1",
    "name": "hillstrom",
    "recordcount": 64000,
    "fieldcount": 3,
    "fields": [
        {
            "name": "recency",
            "type": "integer",
            "role": "independent",
            "tags": [],
            "stats": {
                "nnulls": 0,
                "nuniques": 12,
                "min": 1,
                "max": 12,
                "mean": 5.763734375
            }
        },
        {
            "name": "history_segment",
            "type": "string",
            "role": "independent",
            "tags": [
                "ordinal"
            ],
            "stats": {
                "nnulls": 0,
                "nuniques": 7
            },
            "values": [
                "1) $0 - $100",
                "2) $100 - $200",
                "3) $200 - $350",
                "4) $350 - $500",
                "5) $500 - $750",
                "6) $750 - $1,000",
                "7) $1,000 +"
            ]
        },
        {
            "name": "conversion",
            "type": "boolean",
            "role": "dependent",
            "tags": [],
            "stats": {
                "nnulls": 0,
                "nuniques": 2,
                "min": 0,
                "max": 1,
                "mean": 0.00903125
            }
        },
```


(continues on next page)

(continued from previous page)

```
    ],
    "data": {
        "flatfile": {
            "name": "hillstrom3.csv",
            "format": {
                "separator": ",",
                "quote": "\"",
                "escape": "\\",
                "nullmarker": "",
                "headerrowcount": 1
            }
        }
    }
}
```

Note that not all of the data shown here is required for every PMMIF file.

1.2 Validity and Well-Formedness

PMMIF borrows a useful pair of notions from XML, namely those of *well-formedness* and *validity*.

A PMMIF (`.pmm`) file is well-formed if it conforms syntactically to this definition, i.e. if it is valid JSON, in UTF-8, contains all the required elements and the types and structures of all the elements are correct. To be well-formed, an PMMIF file should also be internally consistent; for example, if a number of fields is specified, it should be the same as the number of fields actually described in the file. More generally, a `.pmm` file is well-formed if there are no errors that can be detected without looking at the datafiles it describes.

By contrast, an PMMIF file is valid only if the datafiles it refers to exist and are consistent with the metadata in the `.pmm` file.

1.3 Extensibility

Extra keys may be added to any of the dictionaries used to define PMMIF provided those keys begin with a capital letter. This allows users to embed arbitrary extra information in a `.pmm` file without significantly compromising the ability of PMMIF checkers to detect errors such as mis-typed keys.

1.4 Generation

CHAPTER 2

Top-Level Structure

Version 0.1 of the PMMIF format is only expressed as JSON as a dictionary with nested elements. Some elements are mandatory, while others are optional. The order of keys does not matter. Indentation and layout of the JSON PMMIF file does not matter, but it is strongly recommended that it is formatted using indentation of four spaces and separated elements on separate lines, as shown below, to aid human readability.

All text elements are case-sensitive.

The PMMIF file must be encoded as UTF-8 and must have the extension `.pmm`, and the flat file must also be encoded as UTF-8.

2.1 Required Top-Level Keys

- `pmmversion` (string): the version of PMMIF to which the file conforms
- `recordcount` (integer): the number of records in the dataset
- `fields` (list): a list of field descriptors as specified below

2.2 Optional Top-Level Keys

- `data` (string): information about the source and format of the data described in the data dictionary, most commonly a flat file
- `name` (string): a name for the dataset
- `description` (string): a description of the dataset (string)
- `contributor` (string): the contributor of the dataset
- `fieldcount` (integer): the number of fields. Should match the length of the field list if specified.
- `data` (Data description object): information about a flat file ("CSV file") containing the data.

CHAPTER 3

Field Descriptions

Fields are described by a dictionary with some required and some optional keys. provision of all relevant keys is recommended, especially summary statistics such as min, man, nullcount, uniquevaluecount etc. is strongly recommnded.

3.1 Required Keys for Field Descriptions

- `name` (string): the name of the field. PMMIF does not place any restrictions on the field names, but it is recommended that they always start with a letter and contain only letters, numbers and underscores for maximum cross-system usablity. Field Names are case sensitive, but it is recommended that multiple fields with names that differ only in capitalization are not used, again to maximize cross-system compatibility.
- `type` (string): the type of the field. PMMIF recognizes the following field types:
  - `boolean` — values that are either `true` or `false`
  - `integer` — integer values, possibly signed
  - `real` — floating-point values
  - `string` — string values, encoded as UTF-8
  - `datestamp` — a date-and-time stamp value, with or without a time zone, in a format to be specified with another key, `format`, that is required when the type is `datestamp`.

  These strings are the only allowed values for field types.
- `role` (string): the role that the field plays in modelling. PMMIF requires that all fields be classified in one of the following ways to make clear its role in modelling:
  - `independent` — a value that can be used in the model as an input for making predictions (also known as a *left-hand* or a *predictor* variable)
  - `dependent` — a value to be predicted by the model (also known as an *outcome*, an *objective*, a *target* or a *right-hand* variable).

- `treatment` — a variable that describes which of a number of treatments a customer, patient or user (record) received. The values are not mandated, but it is recommended that one treatment be nominated as the control, using the `control` key. Treatment fields may be integers, strings or booleans; if not specified, 0, False, "c", "control" (in any case) will be preferred as control indicators.
- `weight` — some kind of weight field, such as a case weight.
- `validation` — an indicator of a fixed set of partitions of the data into different cross-validation segments. No specific interpretation is put on the segments by PMMIF, though this can certainly be specified in notes.
- `auxiliary` — some other kind of field to be used as none of the above, but perhaps to be used; for example, this might be a customer value field, in cases where this is to be used neither for prediction nor as an outcome, but might be used in ROI calculation, for example.
- `ignore` – a field that should be discarded.

3.2 Optional Keys for Field Descriptions

There are various kinds of option keys that we may distinguish:

3.2.1 Clarification Keys

- **`tags`. Tags provide extra information afbout how to use or interpret the data in a field. Recognized tag values are:**
  - `categorical`: this applies to a string field or integer fields and indicates that the different values represent distinct categories that are not ordered but which can sensibly be used for analysis.
  - `ordinal`: for string fields, this indicates that the strings represent some kind of rank. In this case, the `values` key should always be specified and should list the values in rank order, from low to high.
  - `unique`: this indicates that there are (should be) no duplicate values in the field
  - `maximize` or `minimize`: While the goal of predictive modelling is typically to fit the data, it is often the case that the records of interest are either the higher or lower values, and for reporting, it is often useful to know which. A maximize tag means that the high values are of primary interest (e.g. for a response problem. Correspondingly, `minimize` means that the low values are of more interest (e.g. for example, when customer attrition is marked as a 1).
- `values`.

3.2.2 Summary Statistics

- `stats`

3.2.3 Annotations

- `longname`
- `description`

CHAPTER 4

Data Description

PMM is primarily for describing metadata associated with datasets. However, it is also useful for it to be able to describe how data is stored. The data section allows this.

4.1 Optional key: `flatfile`

This describes data stored in a flat file (often referred to generically as "CSV" files, originally standing for *comma-separated file*, but now sometimes used even for files with other separators).

4.1.1 name

The name describes where to find the data on disk. It can be any string, and can be a full path (e.g. `"/usr/local/data/hillstrom.txt"`) or a relative path (such as (`"hillstrom.txt"`)).

4.1.2 format

This describes information about how the data is stored in the flat file.

There are three mandatory keys:

- `encoding` (string): `"UTF-8"` is recommended.
- `separator` (string): Usually a single character, often `","`.
- `headerrowcount` (integer): number of rows in the flat file before the data (most commonly containing field names).

There are also optional keys:

- **`quote` (string): Usually empty (`""`),** a single quote (`"'"`) or a double quote (`"\""`). There is no provision for multiple quote characters at this stage. It is recommended that flat-file readers based on PMM interpret always accept unquoted strings even if a quote character is present.

- `escape` (string): Usually backslash (`"\\"`). The escape character is used primarily to all the quote character and itself to be included in strings by preceding them with the escape character. If set to the empty string, there is no escaping.

  Regardless of the setting of escape, flat file readers may choose to accept escape sequences for special characters, especially `\n` for newline.

- `nullmarker` (string): This is used to describe how nulls (missing values) are listed in the file. If not specifed at all, null values are not allowed in the file. Often set to the empty string (`""`) or a marker such as `NULL`.

- `dateformat` (string): This describes the datestamp format used in the file. When writing data, it is normally preferable to use the same format for all datestamp fields. However, individual fields can also specify a date format, which will override this overall setting. This is necessary if a file already contains dates in mixed formats.

4.1.3 Future extensions

- End-of-line nulls (cf. excel)
- Bad encoding handling?
- Byte-order markers
- Strictness specifiers
- Literal newlines in files

CHAPTER 5

Indices and tables

- genindex
- modindex
- search