Gender Prediction Methods Based on First Names with genderizeR

by Kamil Wais

Abstract In recent years, there has been increased interest in methods for gender prediction based on first names that employ various open data sources. These methods have applications from bibliometric studies to customizing commercial offers for web users. Analysis of gender disparities in science based on such methods are published in the most prestigious journals, although they could be improved by choosing the most suited prediction method with optimal parameters and performing validation studies using the best data source for a given purpose. There is also a need to monitor and report how well a given prediction method works in comparison to others. In this paper, the author recommends a set of tools (including one dedicated to gender prediction, the R package called **genderizeR**), data sources (including the genderize.io API), and metrics that could be fully reproduced and tested in order to choose the optimal approach suitable for different gender analyses.

Introduction

The purpose of the **genderizeR** package and this paper is to provide tools and methods for accurate classification of various types of character strings into gender categories. An increased number of studies require gender identification, as for example, biographical research, when we want to know what is the gender of article authors and we do not have explicit gender data (Larivière et al., 2013b; West et al., 2013; Blevins and Mullen, 2015). Predicting gender of customers for marketing purposes can serve as an example from outside the academia. The **genderizeR** package makes it possible to predict gender related to a character string without knowing which term in the string is in fact a given name. Moreover, the package provides convenient built-in tools for assessing different kinds of error rates specific to gender prediction.

One of the purposes of this paper is to argue that while using informal, crowd-sourced and not widely recognized data sources, one can achieve high gender prediction efficiency comparable to other recognized gender data sources. Moreover, this effect can be obtained with less efforts and higher automatization. The paper explains how we can train models for gender prediction and how we can evaluate those models. There is also a comparison and evaluation of different approaches to gender predictions from other studies.

There have been several approaches proposed for gender prediction based on first names. Some of these methods were used in bibliometrics studies that were published in prestigious scientific journals: Larivière et al. (2013b) or West et al. (2013). One of the goals of this study is to compare the efficiency and accuracy of gender prediction methods used in the mentioned studies and consider a new approach proposed in this paper, which is easier in implementation and usage and yields outcomes comparable or, in some situations, even better than other methods.

For the purpose of method comparison, two methods were chosen from the studies: *The Role of Gender in Scholarly Authorship* (West et al., 2013) and the *Supplementary Information to Global Gender Disparities in Science* (Larivière et al., 2013a), which is the methodological appendix to *Bibliometrics: Global Gender Disparities in Science* (Larivière et al., 2013b). In these studies, authors were predicting and analysing the gender of *authorships*. The instance of an authorship had been defined as *a person and a paper for which the person is designated as a co-author* (West et al., 2013, p. 3) or even simpler as *unique paper-author combination* (Larivière et al., 2013a, p. 6). The sample of authorships also served as a common dataset for comparison of the gender prediction methods (see Section *The comparison of methods*).

Another goal of this study is to find and implement as an R package the most effective gender prediction method that is based on first names and has the following qualities:

- is based on data sources that are available for anyone in a machine-readable format,
- is fully reproducible at any given time,
- is resource effective,
- can be used to update and improve gender predictions over time with new first name data,
- is applicable for multinational studies in various research contexts,
- outperforms other similar methods in terms of accuracy of gender prediction.

Gender prediction accuracy can be defined in different ways, which implies that different metrics can be used to measure it. Later, the author will present a set of metrics that can be used in validation and comparative studies.

In both previously mentioned studies, the authors used open data with information on first names with probable corresponding gender. In order to compare the effectiveness of methods and data sources, we need to be able to reproduce those methods and reuse the same datasets. This is possible, at least to some extent, although there are several issues with the reproducibility of the studies that will be addressed in the following sections.

Review of methods and open data sources

US Census and other data sources

In the first analysed study (Larivière et al., 2013a) authors used both universal and country-specific first name lists. The sources of these data were the *US Census*, *Quebec Census*, *WikiName* portal, multiple *Wikipedia* pages, *top-100-baby-names-search.com* portal, and a few other webpages (Larivière et al., 2013a). In case of some languages, a rule-based approach was used to assign gender based on the *suffix* of a first name. In addition, human coders were used in the study. In the case of 12,828 Chinese names of authors (15.17% of total) with at least 20 papers, the gender was assigned manually by two native speakers from China. They coded the gender of each Chinese name based on their individual knowledge of the Chinese language (Larivière et al., 2013a, p. 4).

The US Census was the primary source of data for gender prediction in the study. In cases where the first name was used for both genders, it was only attributed to a specific gender when it was used at least ten times more frequently for one gender than for the other (Larivière et al., 2013a, p. 2). This rule can be converted to a probability threshold that equals **0.91** or more for the purpose of comparative analysis in this study.

With the methods applied, the paper's authors were able to predict gender for **86%** out of 21 million authorships with full first names from the *Web of Science* database (Larivière et al., 2013a); nevertheless, several major drawbacks of this approach can be identified:

- The presented analysis is **difficult to reproduce**. The full set of first names with corresponding gender data used in this study is not available in machine-readable format, and some data sources even ceased to exist. For example, the *wiki.name.com* portal is not accessible any more (assessed on January 16, 2015).
- The manual coding of gender by humans is **neither fast nor cost-effective**. Moreover, it could be **not reliable enough**, as in the paper there was no information about manual coding accuracy and inter-coder reliability.
- The sources of the data are **not in easily readable machine formats** with the significant exception of the data from the US Census, which can be obtained as text files (United States Census Bureau, 2015a) or as a data directly from the R package **qdap** (Rinker, 2013). In order to use other gender data, they need to be web-scraped and parsed. Moreover, not all *Wikipedia* webpages with country-specific first names have a standardised HTML structure that can be easily utilised in parsing HTML code.
- In this mixed data source approach the **confidence threshold cannot be easily changed**, and researchers are only able to use predefined name categories, such as male, female and unisex. The **category unisex is not very informative**, as it means that we predict that the gender can be either female or male with an unknown probability. Moreover, such a category can be easily recreated when the probability of being a male or a female is known, given the first name (e.g., we can set the predicted category as unisex for all first names that have a 0.5 probability of being male names). In a case when predicted gender is unisex without additional information, it is impossible to precisely assess the effectiveness of such a gender prediction method.

On the other hand, the main advantages of the described approach are as follows:

- Usage of different techniques and country-specific sources in gender prediction could increase the percentage of different types of items with correctly predicted gender and, above all, **decrease the percentage of items with unpredicted gender**.
- There is a strong possibility that at least some of the open data sources used in the analysis will be updated over time, for example *Wikipedia* webpages, and will **include new first names or infrequently used names** in the future. This is not certain, however, as even the US Census Bureau has not provided a newer first name dataset since the 2000 Census (United States Census Bureau, 2015b).

The article by Larivière et al. (2013a) does not explicitly mention **any recognised prediction accuracy metrics** that can be comparable with other gender prediction methods, although its confusion matrix can be rebuilt from the tables with the data from the validation study (see Section *The comparison of methods*).

US Social Security Administration records

In the second analysed approach (West et al., 2013) authors used *US Social Security Administration* (SSA) records with the **top 1000** first names annually collected for each of the 153 million boys and 143 million girls born in the USA from 1880–2010. The authors also have decided *a priori* to use only those records that have at least a **95%** probability to correctly predict gender based on a first name (West et al., 2013).

Based on this method, the authors were able to predict gender for **73%** out of 3 million authorships with full first names from the JSTOR network dataset (Efron, 1983, pp. 2–3).

The main drawbacks of this approach are:

- Non-US first names and names that do not appear in the top 1000 first names cannot be used for gender prediction, so **the first name dataset is US-specific** and is not comprehensive by its definition.
- The authors of the analysis **utilised limited information of the top 1000 baby first names**, although SSA provided an extended version of this database with baby names that occur at least five times in the years between 1880–2013. That extended dataset has information on about 92 600 unique baby names compared to 6 983 unique baby names in the top 1000 dataset.

The main advantages of this approach are:

- The US SSA **baby first names database is updated every year** and is available for anyone as open data in machine, easy-readable format (Social Security Administration, 2015).
- The method is **fully and easily reproducible**, especially with the use of R packages like **gender** (Mullen, 2014) or **babynames** (Wickham, 2014) where the full baby name data provided by the SSA is included as built-in datasets.

The paper does not report **any gender prediction accuracy metrics** (West et al., 2013).

Social network profiles as gender data source (via the genderize.io API)

The third tested approach is our proposition of gender prediction based on first name and gender data from the *genderize.io* database, which was created by Casper Strømgren (Strømgren, 2015a) in August 2013 and has been regularly updated since. Regular incremental updates are possible due to continuous scanning of public profiles and their gender data in major social networks. The database is continuously growing by processing approximately **from 15 000 to 20 000 social network profiles per day**.

On 24-th of May 2014, the *genderize.io* database contained information on **120 517** terms that at least once had been used as a first name in about half a million social network profiles. In April 2015 there were **212 252** unique terms gathered from about **2 million** social network profiles from **79 countries** in **89 languages** (Strømgren, 2015a).

A quick connection to the *genderize.io* database is possible through its *application programming interface* (API). Since February 2014, database queries via the *genderize.io* API have been restricted to 10 terms per request and 1000 terms per day to prevent server overload. Higher limits are easily available through commercial access plans to the API and enable checking up to 10 000 000 names monthly (Strømgren, 2015b).

As a query term to the database, any character string can be used if it is suspected to be a first name (a simple example of a query: `GET http://api.genderize.io?name=peter`). In response to the query, the API returns a null value when the string is not found in the *genderize.io* database. If the term is found in the database, the API returns several values in *JavaScript Object Notation* (JSON) format: a predicted gender for the first name (male or female) and two numeric values that can be further use to customise the gender prediction. These values are *count* and *probability*, where *count* shows how many social network profiles in the database have been recorded with this particular term as a first name and *probability* shows the proportion of profiles with the first name and the predicted gender (a simple example of API response: `"name":"peter","gender":"male","probability":"1.00","count":4300`) (Strømgren, 2015a). Therefore, we not only know the gender prediction for a given term but also how confident we can be with this particular prediction.

Using the *genderizer.io* database through its API for predicting gender has strong advantages:

- The *genderizer.io* database is **continuously and incrementally growing**, thus we are not only able to reproduce classification results using previously saved API output, but we also could improve our predictions next time using newer API output from a larger, updated, and more comprehensive version of the first name database.

Characteristics of approach	Lariviere et al. 2013b	West et al. 2013	*genderize.io*
main data source	US Census	US SSA	public social profiles
other data sources	yes	no	no
open data	some datasets	yes	limited free access
machine-readable format	some datasets	yes	yes
API connection	no	no	yes
easily reproducible	no	yes	yes
resource effective	no	yes	yes
regular data updates	no	yearly	in real time
known probabilities of gender prediction	available only in the main data source	available	available
global reach	country-specific	country-specific	yes

Table 1: Comparison of the characteristics of different approaches to gender prediction based on first names.

- The communication with the API is **fast, effective, and straightforward**; additionally comprehensive documentation is available (Strømgren, 2015a). Communication through the API can be further simplified with the use of dedicated functions in the **genderizeR** package (Wais, 2016).

The issue that can be clearly seen as disadvantage is the daily limit of free queries through the API (1000 terms per day). Much larger limits are still available through commercial plans with reasonable prices. This kind of commercial model behind the *genderize.io* API has also some advantages in comparison with completely free access to the API. It guarantees stability of the service and constant development of the database, covers costs of the servers, and prevents server overloads due to unrestricted access.

The main criticism of this data source is the reliability of the data. The database behind the *genderize.io* API draws on data from numerous public social media profiles, although neither the exact number of sources nor the total number of profiles have been revealed. Even if a social media portal has a real-name policy implemented, there is no guarantee that at a given time each profile has valid and reliable data in its first name and gender fields. So reliability of the data from a perspective of a single profile is very low. However, it is safe to assume that most people give true information regarding their gender and given name, thus such crowd-sourced data aggregated from many profiles can give reliable information due to the scale of the constantly growing database. The major drawback of this data source is the noise in the data related to their declarative character. While creating a profile, one can submit any character string in a given names field. Even if the user profile is corrected later, that bogus data could already be recorded in the *genderize.io* database. This is the obvious disadvantage compared to other official gender data sources, although the noise can be reduced by setting a higher threshold for profile counts. In this way, we are able to use only those terms for given names and gender data that were also entered in some other social media profiles and therefore seem to be 'confirmed' by other users.

Comparison of approach characteristics

Table 1 shows that the Larivière et al. (2013a) approach is based on a resource-consuming method and lacks some important characteristics like reproducibility. The approach in West et al. (2013) is fully reproducible and enhanced with other important characteristics, but the data source used is US-specific and infrequently updated. Utilising the *genderize.io* database via its fast API, we gain access to a global database of first names that is being continuously updated even at this moment.

The genderizeR package

In order to provide an effective tool for performing and evaluating gender prediction methods, the **genderizeR** package for R has been built. The package enables convenient communication with the *genderize.io* API and has built-in functions for evaluating the effectiveness of gender prediction with metrics specific to this topic.

The package could be used for different tasks:

- for pre-processing text vectors for future gender prediction;
- for connecting with the *genderize.io* database through its API;
- for *genderizing* character strings, which means that the gender is predicted even if we do not indicate which term from the string is the first name. The algorithm assumes that all input terms could be potentially gender indicators and searches for the most credible one;
- for *training* gender prediction algorithms (looking for the optimal combination of *gender* and *probability* parameters from a given set in order to minimise the gender prediction accuracy metric);
- for estimation of different gender prediction accuracy metrics.

There are four main components of the package:

- functions working with the *genderize.io* API (`textPrepare`, `genderizeAPI`, `findGivenNames`);
- functions for training and predicting gender (`genderize`, `genderizeTrain`, `genderizePredict`);
- functions assessing different kinds of gender prediction errors (`classificationErrors`, `genderizeBootstrapError`);
- training datasets (`authorships`, `givenNameDB_authorships`, `titles`, `givenNamesDB_titles`).

The `genderizeBootstrapError` function is based on code from the **sortinghat** package (Ramey, 2013). The function has built-in functionality for parallel processing working directly with functions from the **genderizeR** package (`genderizeTrain` and `genderizePredict`). The parallel processing uses the **parallel** package and its implementation was inspired by Nathan VanHoudnos' scripts (`http://edustatistics.org/nathanvan/setup/mclapply.hack.R`).

The `textPrepare` function for text pre-processing helps to prepare terms for API queries. It utilizes functions from the R packages **stringr** (Wickham, 2012) and **tm** (Feinerer and Hornik, 2014; Feinerer et al., 2008) to perform a series of pre-processing steps (removing special characters, numbers and punctuation; building a vector of unique terms which can be used for creating API queries).

A trivial example of basic package functions

If we use the package for the first time, we need to install it from the *Comprehensive R Archive Network* (CRAN) or from the *GitHub* repository. The last stable version of the package is on CRAN, and the latest development version of the package is available from the GitHub repository "kalimu/genderizeR" (Wais, 2016).

With the `findGivenNames` function we can easily look for first names in the *genderize.io* database. In return, we obtain a data table with the following records: the terms that appear in the database as first names, their predicted gender, probability of such a prediction, and the count of profiles on which the prediction is based . The outcome is alphabetically sorted by the *name* column, and all terms that appear to be first names are lower-cased. Because the outcome is generated from the current state of the *genderize.io* database, to reuse exactly the same outcome later, we should save it locally. This is an important step in reproducible analysis because if we run the same function next time, our output could be based on a larger count of social network profiles and could give us a slightly different data. The `progress = FALSE` argument turns off the progress bar.

```
R> library('genderizeR')
R> findGivenNames(c('Marie', 'Albert', 'Iza', 'Olesia', 'Marcin', 'Andrzej', 'Kamil'),
+                 progress = FALSE)

      name gender probability count
1:  albert   male        0.99   710
2: andrzej   male        0.98    49
3:     iza female        1.00    28
4:   kamil   male        0.99   124
5:  marcin   male        1.00   128
6:   marie female        0.99  2248
7:  olesia female        1.00     4
```

In some cases, we might need to predict the gender of a person based on the full name. It is trivial when we know which term is a first name (or first names) because we can manually extract it and check its gender with the `findGivenNames` function. When we do not know exactly which term in a character vector is a first name, the same function attempts to predict gender by analysing unique terms in the vector.

This way we could predict the gender of an author of an article without explicitly extracting the first name from the full name. For example, one biographical article from the *Web of Science* (WOS)

database records is titled "Marie Sklodowska-Curie (1867–1934)". We know that the author's full name of the article is recorded as "Pascual-Leone Pascual, Ana Ma" (WOS, 2014).

```
R> x <- 'Pascual-Leone Pascual, Ana Ma'
```

We can use the `findGivenNames` function directly on our vector. The function first calls another function from the package, `textPrepare`, which builds a vector of unique terms that are used in queries to *genderize.io* API. This is more effective than checking the same term many times through the API. In the outcome of the `textPrepare` function, we have terms which are at least two characters long, as we can assume that a one-character term could not be a full first name. The text pre-processed tasks that `textPrepare` function can perform are:

- removing single-character terms (like initials),
- removing punctuation,
- removing special characters (like exclamation marks, question marks, hyphens, etc.),
- removing numbers,
- converting all characters to lowercase,
- striping white-spaces.

```
R> textPrepare(x)
[1] "ana"     "leone"   "ma"      "pascual"
```

In the next step, we can check our terms in the *genderize.io* database and store the output for later use.

```
R> (genderDB <- findGivenNames(x, progress = FALSE))

      name gender probability count
1:     ana female        0.99  3535
2:   leone female        0.81    27
3:      ma female        0.62   243
4: pascual   male        1.00    25
```

All four unique terms occur in the *genderize.io* database and could be used as gender indicators, but the *count* value suggests which terms are indeed true first names in our example. The term 'Ana' has the largest *count* value, so it will be used to predict gender for the given author with the `genderize` function.

```
R> genderize(x, genderDB = genderDB, progress = FALSE)

                          text givenName gender genderIndicators
1: Pascual-Leone Pascual, Ana Ma       ana female                4
```

In the final step, we used the previous output as gender information for our terms. We use an output from the `findGivenNames` function, but it could also be a dataset from the US Census or US SSA, although it should be converted to the same format with the same column names.

The `genderize` function output contains our original author's full name in the `text` column, the term that was assumed to be the gender indicative first name (`givenName` column), and the predicted gender (`gender` column). There are also `genderIndicators` values that show how many terms in a `text` were found in our gender dataset. The `givenName` term is the one that won the competition between terms to be a final gender indicator. To find the winning term, we compare the counts for all terms found and choose the one with the maximum value. It is a reasonable way to prevent prediction based on accidental terms.

It should be noted that in cases when a person has a double first name like 'Hans-Peter' the terms 'Hans' and 'Peter' are looked up in the database separately and the gender prediction is still done based on the term with more counts. This is a counter-intuitive solution as the *genderize.io* API can accept queries with double first names. However it simplifies the algorithm and still gives proper predictions even if the double name is not present in the database.

```
R> x <- 'Hans-Peter'
R> (genderDB <- findGivenNames(x, progress = FALSE))

    name gender probability count
1:  hans   male        0.99   425
2: peter   male        1.00  4300
```

```
R> genderize(x, genderDB = genderDB, progress = FALSE)

         text givenName gender genderIndicators
1: Hans-Peter     peter   male                2
```

In the same way, we can predict gender not only from authors' full names but also from larger character strings, for example, titles of the biographical articles. In the next example, we have six such titles in which some person names are mentioned. Half of the articles in this example are about Marie Sklodowska-Curie and half are about Albert Einstein. Let us assume that we do not know which title concerns a female and which a male, and we want to check it automatically.

```
R> x <- c('Marie Sklodowska-Curie (1867-1934) - A person and scientist',
+         'Albert Einstein as a philosopher of science',
+         'The legacy of Albert Einstein (1879-1955)',
+         'Life and work of Marie Sklodowska-Curie and her family',
+         'Marie Sklodowska-Curie (1867-1934)',
+         'Albert Einstein, radical - A political profile')
R> (givenNamesDB <- findGivenNames(x, progress = FALSE))

        name gender probability count
1:    albert   male        0.99   710
2:        as   male        0.89    64
3:  einstein   male        1.00     4
4:    family   male        1.00     1
5:       her female        0.50     8
6:    legacy   male        1.00     2
7:     marie female        0.99  2248
8: political   male        1.00     1
9:       the female        0.50     2
```

We should note that cases are possible where the number of females is the same as the number of males (the probability equals 0.50, and the count is an even number). In our example, such a situation occurs with the *her* term. In such situations, the API returns female as gender prediction, but in the future it will likely be changed to the more valid unisex category.

There are some noise in the gender data obtained from the *genderize.io* API so we should approach the results from the `findGivenNames` function critically. In the above example the terms "as", "the" and "her" seem to be used in some profiles even if they do not stand for a given name. We can deal with such terms by setting the `count` threshold high enough (ex. `count >= 100`) or by using a list of non-acceptable words (*blacklist / stopwords*). The second solution can be implemented to the result of the `textPrepare` and before using `findGivenNames` function. This way we will also reduce the usage of API requests. Using a counts threshold we need to remember that its height should be relative to the growing number of profile data processed by *genderize.io*. However, in our example the noise does not impact the accuracy of the gender predicion.

```
R> genderize(x, genderDB = givenNamesDB, progress = FALSE)

                                                          text
1: Marie Sklodowska-Curie (1867-1934) - A person and scientist
2:                 Albert Einstein as a philosopher of science
3:                   The legacy of Albert Einstein (1879-1955)
4:      Life and work of Marie Sklodowska-Curie and her family
5:                           Marie Sklodowska-Curie (1867-1934)
6:              Albert Einstein, radical - A political profile
   givenName gender genderIndicators
1:     marie female                1
2:    albert   male                3
3:    albert   male                4
4:     marie female                3
5:     marie female                1
6:    albert   male                3
```

We can also set the thresholds for *probability* or *count* values to exclude accidental terms or unisex first names from being considered in a gender prediction. This method can be used to train the prediction algorithm in order to minimise the classification error. However, we need to be careful while setting the thresholds because, if we set them too high, we could unintentionally exclude some true first names and the algorithm will not be able to predict gender, returning only `NA` values.

```
R> genderize(x, givenNamesDB[count > 2000, ], progress = FALSE)

                                                              text
1: Marie Sklodowska-Curie (1867-1934) - A person and scientist
2:                 Albert Einstein as a philosopher of science
3:                   The legacy of Albert Einstein (1879-1955)
4:      Life and work of Marie Sklodowska-Curie and her family
5:                           Marie Sklodowska-Curie (1867-1934)
6:              Albert Einstein, radical - A political profile
   givenName gender genderIndicators
1:     marie female                1
2:        NA     NA                0
3:        NA     NA                0
4:     marie female                1
5:     marie female                1
6:        NA     NA                0
```

The communication with the *genderize.io* API is based on UTF-8 encoding. We can also make use of a specific locale.

```
R> Sys.setlocale("LC_ALL", "Polish")
R> (x <- "Róża")
[1] "Róża"

R> (xp <- textPrepare(x))
[1] "róża"

R> findGivenNames(xp, progress = FALSE)
#    name gender probability count
# 1: róża female        0.89    28
```

If the `findGivenNames` function stops due to the API free or commercial plan limits the results from already performed queries are returned and can be saved as an object.

```
R> xPrepared <- textPrepare(authorships[['value']][1:1200])
R> givenNames_part1 <- findGivenNames(xPrepared)

Terms checked: 10/86. First names found: 4.          |   0%
Terms checked: 20/86. First names found: 7.          |  11%
Terms checked: 30/86. First names found: 12.         |  22%
Terms checked: 40/86. First names found: 17.         |  33%
Terms checked: 50/86. First names found: 22.         |  44%
Terms checked: 60/86. First names found: 25.         |  56%
  |================================                  |  67%
 Client error: (429) Too Many Requests (RFC 6585)
 Request limit reached

The API queries stopped at 57 term.

Warning messages:
1: In genderizeAPI(termsQuery, apikey = apikey, ssl.verifypeer = ssl.verifypeer) :
   You have used all available requests in this subscription plan.
2: In findGivenNames(xPrepared) : The API queries stopped.
```

Moreover the function reports an index of the last term which has been successfully queried before reaching the API limit. This enables us to start the API queries from that term when the API plan resets the next day.

```
R> givenNames_part2 <- findGivenNames(xPrepared[57:NROW(xPrepared)])
```

Finally, we can bind all parts together.

```
R> givenNames <- rbind(givenNames_part1, givenNames_part2)
```

Sample datasets

In the **genderizeR** package we have two sample datasets: `authorships` and `titles`. Both datasets contain data from a simple random sample of biographical articles from the WOS database records of articles of *biographical-items* or *items-about-individual* type from all fields of study, published from 1945 to 2014. The sample was drawn in December 2014. The first dataset `authorships` contains 2 641 authorships of 2 000 randomly sampled records of biographical articles.

Part of the authors in this dataset were successfully identified and manually coded as females or males (`genderCoded` column). Human coders based their coding decisions on results from Internet queries regarding full names of the authors and their affiliations, biographies, mentions in the press and photos. If a full name of an author was unavailable the gender was coded as `noname`; if a full name was available but the coder was not able to code the author's gender with a high degree of certainty — the record was coded as `unknown`.

```
R> tail(authorships[, c(4, 5)])
```

```
             value genderCoded
2636  Morison, Ian        male
2637 Hughes, David        male
2638 Higson, Roger        male
2639    CONDON, HA      noname
2640  GILCHRIST, E      noname
2641     Haury, LR      noname
```

The second dataset `titles` contains 1 190 titles of biographical articles, which also were manually coded as female or male.

```
R> tail(titles)[-3, ]
```

```
                                          title genderCoded
1:   Yuri A. Chizmadzhev (to the 80th anniversary)        male
2:           Yurko Duda, a physicist like few ones        male
3: Zhores Ivanovich Alferov (on his 80th birthday)        male
4:          Zongluo Luo, a Chinese Haigui in 1930s        male
5:                                    lynn seymour      female
```

These datasets can be used to assess the efficiency and accuracy of gender prediction for different prediction methods and related data sources.

For reproducibility purposes, in the package there are two additional datasets corresponding to the previous two: `givenNamesDB_authorships` and `givenNamesDB_titles`. Both are outputs of the `findGivenNames` function, which was used on authorship and title vectors on December 26, 2014. These datasets can be used for gender prediction without the necessity of connecting to the *genderize.io* database, and to provide reproducible outcomes. Using API queries is also possible, although it will probably give slightly different outputs due to the first name data updates in the database.

Selecting prediction parameters

Knowing that each first name record from the *genderize.io* database has been assigned a probability of predicted gender, we can utilise these pieces of information to strengthen the confidence of gender prediction. Moreover, for each record, the number of instances of social network profiles that were used to calculate such a probability is known. Thus, we can set our own thresholds of *counts* and *probabilities* when we need it, and choose settings that can optimise chosen prediction efficiency metrics.

Metrics of gender prediction efficiency

To assess the efficiency of gender prediction, we need to agree on the set of efficiency indicators that are adequate to this particular classification problem. We can calculate some of the indicators with the help of `classificationErrors` function that are based on a prediction confusion matrix – as an example below of randomly generated samples of labels and predictions.

```
R> set.seed(238)
R> labels <- sample(c('male', 'female', 'unknown'), size = 100, replace = TRUE)
R> predictions <- sample(c('male', 'female', NA), size = 100, replace = TRUE)
```

```
R> indicators <- classificationErrors(labels, predictions)
R> indicators[['confMatrix']] # confusion matrix for the generated sample

          predictions
labels     female male <NA>
  female       12   10    4
  male          7   10   12
  unknown      16   13   16
  <NA>          0    0    0
```

For the confusion matrix, we could calculate a standard classification error rate (one minus sum of the diagonal divided by sum of total), but this is not directly applicable in this specific classification problem; some titles were manually coded as `unknown` gender if they do not contain a first name or when we could not verify the gender of the person mentioned in a title. In this case, we should not blame the algorithm for wrongly predicting gender, considering the human coder could not do it either. Instead, we can calculate the **coded error rate** on the confusion matrix with manual female and male codes only.

```
R> # errorCoded <- (7 + 10 + 4 + 12) / (12 + 10 + 4 + 7 + 10 + 12)
R> unlist(indicators['errorCoded'])

errorCoded
       0.6
```

The advantage of such an error rate calculation is that it incorporates within itself all items with unpredicted gender (`<NA>`). Therefore, if the algorithm is unable to predict gender, we treat it as a prediction error and thus penalise our prediction efficiency metric.

Another possibility is to calculate the coded error rate but only for a matrix with automatically predicted gender (without any `NA` values).

```
R> # errorCodedWithoutNA <- (7 + 10) / ( 12 + 10 + 7 + 10)
R> unlist(indicators['errorCodedWithoutNA'])

errorCodedWithoutNA
          0.4358974
```

Such a **coded error rate without `NA` values** indicates how well the algorithm predicts gender, provided that prediction is possible. With a high threshold of gender probability, we increase the prediction accuracy, but we also risk that for many first names the algorithm will not be able to predict their gender. Therefore, the cost of high prediction accuracy can be measured as the increase of the proportion of items with unpredicted gender.

```
R> # naCoded <- (4 + 12) / ( 12 + 10 + 4 + 7 + 10 + 12)
R> unlist(indicators['naCoded'])

  naCoded
0.2909091
```

There is another problem-specific error that we can calculate from the confusion matrix. It can be called the **gender bias error rate**, and can be calculated as a difference between the number of items manually labelled as female but automatically classified as male and the number of items manually classified as male but automatically classified as female, and all that are divided by the number of all items labelled or classified as female or male.

```
R> # errorGenderBias <- (7 - 10) / (12 + 10 + 7 + 10)
R> unlist(indicators['errorGenderBias'])

errorGenderBias
    -0.07692308
```

We can interpret the sign of such an indicator as a direction of the bias in gender prediction. Negative values suggest that more females are incorrectly classified as males than males as females, hence the predicted proportion of females could be slightly overestimated. If the value is positive, the proportion of females are underestimated. When the bias is close to 0, we have the case where nearly the same numbers of both female and male items are classified wrongly as males and females, respectively. Therefore, in the trivial example, if we have a sample of 10 males and 10 females and we

Prediction indicator	What items are taken into account?
`errorCoded`: Coded Error Rate	Items with manually coded female and male labels; items with predicted female and male labels, or with unpredicted gender.
`naCoded`: Proportion of Items with Unpredicted Gender	Items with manually coded female and male labels; items with predicted female and male labels, or with unpredicted gender.
`errorCodedWithoutNA`: Coded Error Rate without `NA` Values	Items with manually coded female and male labels, and with predicted female and male labels only.
`errorGenderBias`: Gender Bias Error Rate	Items with manually coded female and male labels, and with predicted female and male labels only.

Table 2: Comparison of selected gender prediction indicators.

incorrectly classify five males as females and five females as males, we will still have a proportion of 50% of female items in the sample, and the bias error is 0.

A brief comparison of the described indicators is shown in Table 2. The table indicates that we always focus on metrics based on manually coded gender labels with the exception of two indicators, where we consider items with unpredicted gender as well.

Depending on the research goals, we can utilise one or more of these metrics, for example:

- in a scenario where the goal is to predict gender for as many items as possible, we can try to minimise the **coded error rate**,
- in a scenario where the prediction accuracy is more important than the percentage of items with unpredicted gender, we can try to minimise the **coded error rate without `NA` values**,
- in a scenario where we primarily want to accurately estimate the proportion of males or females, we can try to minimise the **gender bias error rate**.

Different decisions can be made corresponding to different research needs. We can try to minimise all these indicators at the same time and look for their optimal values based on our research questions.

Case study 1: Authorships

In order to establish optimal values of gender prediction indicators for the authorship sample dataset, we can manipulate the thresholds of `probability` and `count` values. In order to do so, we can use the `genderizeTrain` function with values that we prefer to be considered in the *training* of the algorithm as the function arguments `probs` and `counts`. In this case, we can utilise some typical probability values used for gender prediction together with different values of counts that will give noticeable patterns of error rates for parameter combinations.

```
R> probs <- c(0.5, 0.7, 0.8, 0.9, 0.95, 0.97, 0.98, 0.99, 1)
R> counts <- c(1, 10, 100)
R> authorshipsGrid <-
+    genderizeTrain(# parallel = TRUE,
+                   x = authorships$value,
+                   y = authorships$genderCoded,
+                   givenNamesDB = givenNamesDB_authorships,
+                   probs = probs,
+                   counts = counts)
```

The `genderizeTrain` function first creates the grid of parameters for all unique combinations of `probs` and `counts` values. Then, prediction is done based on `givenNamesDB` data trimmed by `probs` or `counts` parameters. As the output, we attain gender prediction defectiveness indicators for all combinations of parameters (27 in our example below).

```
R> authorshipsGrid
```

```
   prob count errorCoded errorCodedWithoutNA    naCoded errorGenderBias
 1: 0.50     1 0.07093822          0.03791469 0.03432494     0.014218009
 2: 0.70     1 0.08466819          0.03147700 0.05491991     0.007263923
 3: 0.80     1 0.10983982          0.03233831 0.08009153     0.012437811
 4: 0.90     1 0.11899314          0.03022670 0.09153318     0.015113350
 5: 0.95     1 0.13272311          0.02820513 0.10755149     0.012820513
 6: 0.97     1 0.14645309          0.02610966 0.12356979     0.010443864
 7: 0.98     1 0.15560641          0.02638522 0.13272311     0.010554090
 8: 0.99     1 0.18306636          0.02724796 0.16018307     0.005449591
 9: 1.00     1 0.27459954          0.03353659 0.24942792    -0.003048780
10: 0.50    10 0.12128146          0.03759398 0.08695652     0.017543860
11: 0.70    10 0.13958810          0.02842377 0.11441648     0.007751938
12: 0.80    10 0.16247140          0.02917772 0.13729977     0.013262599
13: 0.90    10 0.16933638          0.02680965 0.14645309     0.016085791
14: 0.95    10 0.18535469          0.02465753 0.16475973     0.013698630
15: 0.97    10 0.19908467          0.02234637 0.18077803     0.011173184
16: 0.98    10 0.20823799          0.02259887 0.18993135     0.011299435
17: 0.99    10 0.23569794          0.02339181 0.21739130     0.005847953
18: 1.00    10 0.33180778          0.02666667 0.31350114     0.000000000
19: 0.50   100 0.27459954          0.03058104 0.25171625     0.012232416
20: 0.70   100 0.29061785          0.02821317 0.27002288     0.009404389
21: 0.80   100 0.30892449          0.02893891 0.28832952     0.016077170
22: 0.90   100 0.31350114          0.02912621 0.29290618     0.016181230
23: 0.95   100 0.32036613          0.02941176 0.29977117     0.016339869
24: 0.97   100 0.32951945          0.02657807 0.31121281     0.013289037
25: 0.98   100 0.33638444          0.02684564 0.31807780     0.013422819
26: 0.99   100 0.36613272          0.02807018 0.34782609     0.007017544
27: 1.00   100 0.45995423          0.02880658 0.44393593     0.004115226
```

We could also visualise the prediction effectiveness of the parameter grid on a scatterplot and look for optimal values of indicators (Figure 1). In this case, we will use only a subset of probability values (0.5, 0.8, 0.95, 0.98, 1) in order to keep the scatterplot clear.

If the goal is to minimise the coded error rate, we should set the threshold values as low as possible (`prob = 0.5` and `count = 1`).

```
R> authorshipsGrid[authorshipsGrid$errorCoded == min(authorshipsGrid$errorCoded), ]

  prob count errorCoded errorCodedWithoutNA    naCoded errorGenderBias
1:  0.5     1 0.07093822          0.03791469 0.03432494      0.01421801
```

However, if the goal is to have the lowest gender bias error rate, we should set the threshold values as `prob = 0.97` and `count = 10`. However, in this way we increase our proportion of items with unpredicted gender from less then 4% to more than 18%.

```
R> authorshipsGrid[authorshipsGrid$errorGenderBias ==
+                    min(abs(authorshipsGrid$errorGenderBias)), ]

   prob count errorCoded errorCodedWithoutNA   naCoded errorGenderBias
1: 0.97    10  0.1990847          0.02234637 0.180778      0.01117318
```

Case study 2: Titles of biographical articles

As in the first case study, we can train the algorithm on the second dataset with titles of biographical articles, using a wider set of values for both `probs` and `counts` parameters. In the case of a huge parameter grid, we can also try to run the `genderizeTrain` function in parallel version (see `parallel` argument).

```
R> probs <- seq(from = 0.5, to = 0.9, by = 0.05)
R> probs <- c(probs, seq(from = 0.91, to = 1, by = 0.01))
R> counts <- seq(from = 1, to = 16, by = 1)
R> titlesGrid <-
+    genderizeTrain(# parallel = TRUE,
+                   x = titles$title,
+                   y = titles$genderCoded,
+                   givenNamesDB = givenNamesDB_titles,
```

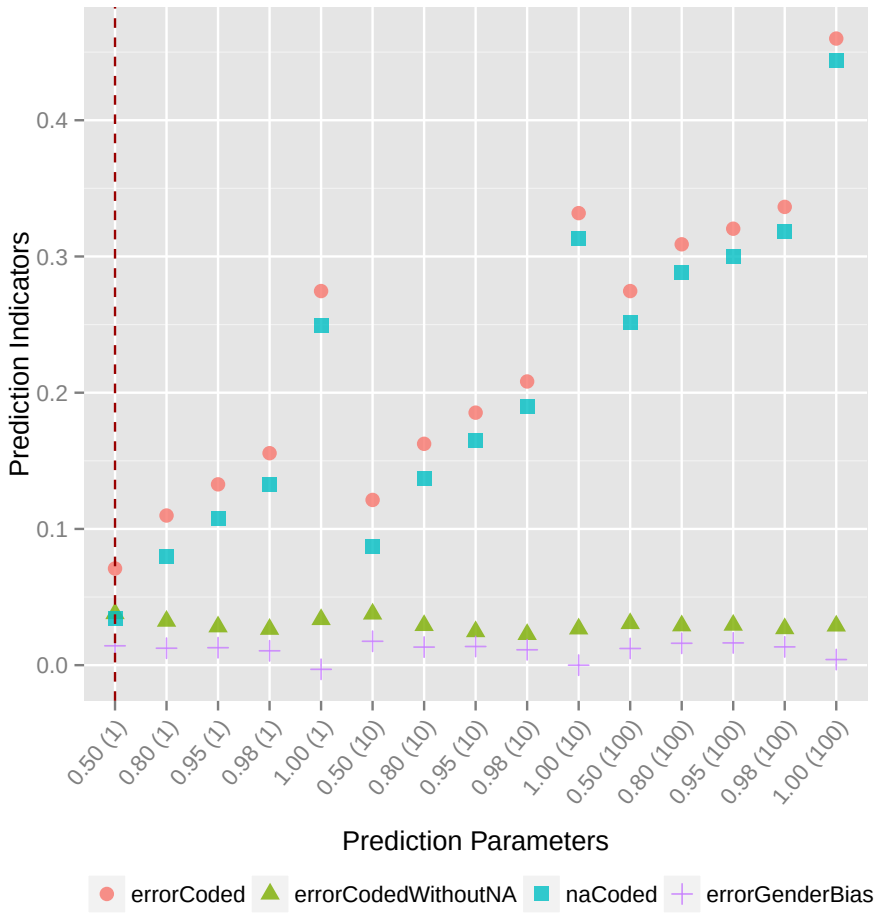

Figure 1: Effectiveness of gender prediction for given parameters from the authorship sample. The optimal effectiveness, when all indicators have the lowest values possible, seems to be achievable with probability = 0.50 and count = 1 and is marked on the plot as a dashed red line. The proportion of items with unpredicted gender (`naCoded`) increases with the increase of parameter values. The lowest gender bias error rate (`errorGenderBias`) can be achieved by setting the probability parameter to 1, although it will greatly increase the proportion of unpredicted items.

```
+                  probs = probs,
+                  counts = counts)
```

Because the scatterplot for all parameter combinations would be very unclear, we have visualised only a subset of count values: 1, 4, 7 (Figure 2). The trends and patterns that emerged from such a simplified visualisation are similar to other combinations of parameters.

Parameter values `prob = 0.70` and `count = 1` minimised coded error rate for this dataset produced a reasonable low gender bias error rate. We can also achieve lower gender bias error rate; however, it comes with an increased proportion of unpredicted items.

```
R> titlesGrid[titlesGrid$errorCoded == min(titlesGrid$errorCoded), ]

   prob count errorCoded errorCodedWithoutNA    naCoded errorGenderBias
1:  0.7     1  0.1004367          0.06533575 0.03755459      0.02540835
R> titlesGrid[titlesGrid$errorGenderBias ==
+             min(abs(titlesGrid$errorGenderBias)), ]

   prob count errorCoded errorCodedWithoutNA   naCoded errorGenderBias
1:    1    15  0.2917031          0.04023669 0.2620087     0.002366864
2:    1    16  0.2917031          0.04023669 0.2620087     0.002366864
```

The final decision on which set of parameter values to use can be subjective and is dependent on the importance of each indicator in a given analysis.

Further estimation of prediction accuracy

For further estimation of gender prediction accuracy we can use cross-validation or bootstrap methods like *Leave-One-Out Bootstrap* (LOO) (Efron, 1983) and *.632+ Rule* (Efron and Tibshirani, 1997) as well as *Receiver Operating Characteristic* (ROC) and *Area Under Curve* (AUC) (Fawcett, 2003) or finally *Brier Score* (Brier, 1950) as one of the most popular performance metrics for probabilistic classifiers.

Bootstrapping

Up to this moment, we have only relied on the so called *apparent error rate*, which is *the observed inaccuracy of the fitted model to the original data points*. However, the apparent error rate usually underestimates the true error rate and gives a falsely optimistic picture of the model's true accuracy (Efron, 1986). To address this issue, we can calculate the *Leave-One-Out (LOO) bootstrap error rate* proposed by Efron (1983).

In the LOO bootstrap procedure, we generate some number of bootstrap samples of the size of the original sample. On each bootstrap sample, we can train our prediction algorithm by minimising the classification error. Later, we can use optimal parameters to predict gender for items that have not been sampled in this particular bootstrap sample. This way, we avoid testing a prediction model on the items used for choosing the prediction parameters.

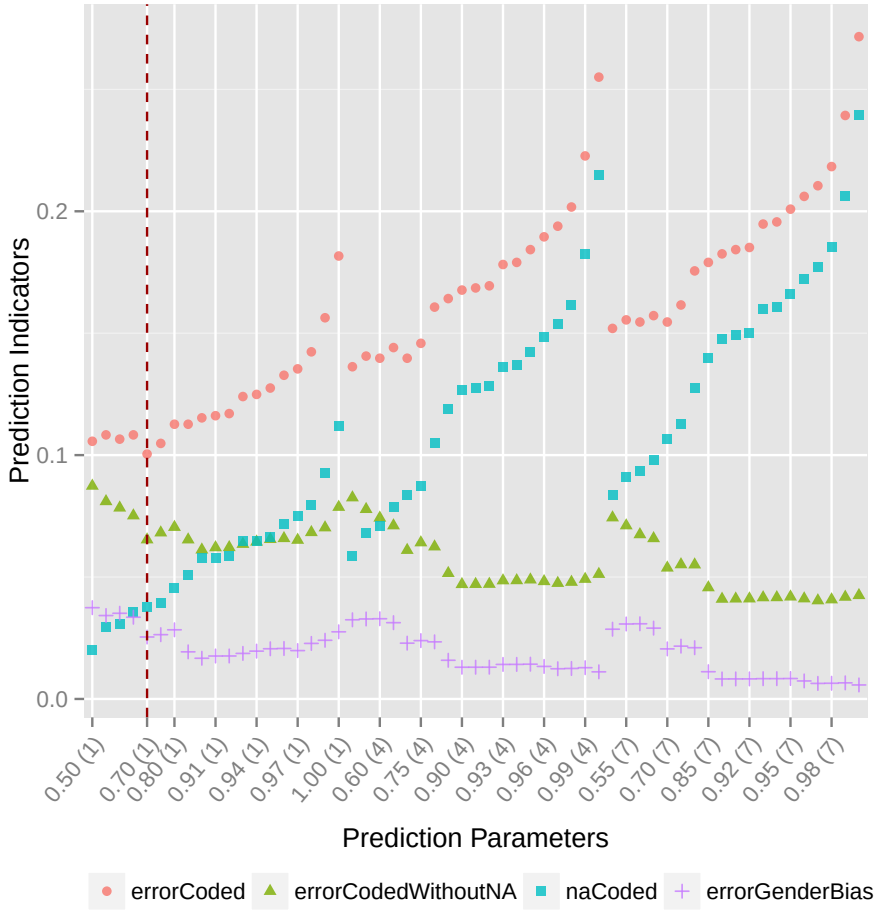

Figure 2: Effectiveness of gender prediction for given parameters from the title sample. The lowest `errorCoded` is achieved with probability = 0.70 and count = 1 (marked on the plot as a dashed red line), which also yields resonable low values of other error rates. The proportion of items with unpredicted gender (`naCoded`) increases periodically with the increase of the parameter values. The gender bias error rate (`errorGenderBias`) could be minimised at the cost of increasing the proportion of items with unpredicted gender. The final decision on which set of parameter values to use can be subjective and is dependent on the importance of each indicator in a given analysis.

As Jiang and Simon (2007) summarised, the LOO bootstrap is a smoothed version of the LOO cross-validation. Bootstrap samples differentiate among each other more than the original LOO sets. Moreover, for each item, the LOO bootstrap method averages the errors from the multiple predictions made on the bootstrap samples. As a result, the LOO bootstrap estimate has a much smaller variability than the LOO cross-validation estimate.

On the other hand, in contrast to low bias cross-validation, the LOO bootstrap in some cases presents noticeable bias and tends to overestimate the true prediction error. The *.632+* estimator was proposed as a remedy to deal with this problem (Efron and Tibshirani, 1997).

In gender prediction, we can use these bootstrap methods, as they are implemented in **genderizeR** package in the `genderizeBootstrapError` function.

We can still train our prediction algorithm on a large grid of parameters, or we can focus on combinations that we suspect could yield different outcomes on different bootstrap samples. In our `titles` dataset, we found that parameters `prob >= 0.70` and `count >= 1` give us the lowest apparent error rate on coded items, but it is reasonable to suspect that the combination of `prob >= 0.50` and `count >= 1` also could give us the lowest error rate on some random samples.

```
R> counts <- 1
R> probs <- c(0.5, 0.7)
R> set.seed(42)
R> bootstrapErrors <- genderizeBootstrapError(
+    # parallel = TRUE,
+    x = titles[titles$genderCoded %in% c('female', 'male')]$title,
+    y = titles[titles$genderCoded %in% c('female','male')]$genderCoded,
+    givenNamesDB = givenNamesDB_titles,
+    probs = probs,
+    counts = counts,
+    num_bootstraps = 50)
R> t(as.data.frame(bootstrapErrors))

                     [,1]
apparent         0.1004367
loo_boot         0.1037184
errorRate632plus 0.1025251
```

All bootstrap errors are calculated only on items with gender labels and predictions. The `apparent` value presents the underestimated *apparent error rate*; the `loo_boot` value provides the overestimated *LOO bootstrap error rate* and the `errorRate632plus` value gives *.632+* estimator that provides the best estimation of prediction error rate.

ROC and AUC

In order to plot the ROC curve or calculate AUC (Fawcett, 2003) for our predictions on the `title` sample, we can use the R package **ROCR** (Sing et al., 2005), but the datasets need to be prepared first. We need to have a data frame that shows which title has manually coded female (or male) gender types with a column of corresponding probabilities predicting these genders. To do that, we need to combine the original dataset with the prediction and corresponding gender data.

```
R> genderizedTitles_output <-  genderize(x = titles[['title']],
+                                        genderDB = givenNamesDB_titles,
+                                        progress = FALSE)
R> genderizedTitles <- cbind(as.data.frame(titles),
+                            as.data.frame(genderizedTitles_output))
R> genderizedTitles <-
+    left_join(x = genderizedTitles[, names(genderizedTitles) != 'gender'],
+              y = givenNamesDB_titles, by = c("givenName" = "name"))
R> example <- rownames(genderizedTitles[c(3, 9, 25, 27, 29, 37), ])
R> genderizedTitles %>%
+    dplyr::select(title, genderCoded, givenName,
+    probability, count, gender) %>%
+    filter(rownames(.) %in% example)

                                                                  title
1                      (Jacqueline) Nancy Mary Adams, CBE, QSO 1926-2007
2                                2005 R W P King Award - Robert J. Adams
```

```
3                                    A master potter (Josiah Wedgwood)
4 A musician without retirement - Claus Bantzer ceases and makes it again
5                   A pioneer in the world of advertising, Armando Testa
6                                    A tribute to Jean-Michel Quinodoz
  genderCoded givenName probability  count gender
1      female      mary        1.00  54051 female
2        male    robert        1.00 100216   male
3        male    josiah        0.98     43   male
4        male     claus        0.94     81   male
5        male   armando        0.99    329   male
6        male      jean        0.52  26799   male
```

In the next step, we can compute our vector of probabilities that we use to predict a given gender for each item.

```
R> d <- genderizedTitles
R> d[is.na(d[['gender']]), ][['gender']] <- 'unknown'
R> d <- d[d[['genderCoded']] %in% c('female', 'male', 'unknown' ), ]
R> d <- d[d[['gender']]  %in% c('female', 'male', 'unknown' ), ]
R> d$labels <- ifelse(d[['genderCoded']] == 'female', 1, 0)
R> d$pred <- ifelse(d[['gender']] == 'female',
+                   as.numeric(d[['probability']]),
+                   1-as.numeric(d[['probability']]))
R> d %>% dplyr::select(title, genderCoded, labels, pred) %>%
+    filter(rownames(.) %in% example)

                                                                    title
1                      (Jacqueline) Nancy Mary Adams, CBE, QSO 1926-2007
2                                2005 R W P King Award - Robert J. Adams
3                                    A master potter (Josiah Wedgwood)
4 A musician without retirement - Claus Bantzer ceases and makes it again
5                   A pioneer in the world of advertising, Armando Testa
6                                    A tribute to Jean-Michel Quinodoz
  genderCoded labels pred
1      female      1 1.00
2        male      0 0.00
3        male      0 0.02
4        male      0 0.06
5        male      0 0.01
6        male      0 0.48
```

Now we can plot the ROC curve and calculate the AUC.

```
R> library('ROCR')
R> pred <- prediction(labels = d[['labels']], predictions = d[['pred']])
R> perf <- performance(pred, measure = 'tpr', x.measure = 'fpr')
R> # area under the curve (AUC)
R> unlist(performance(pred, measure = 'auc')@y.values)

[1] 0.9186008
```

The AUC value is high (0.92), so we can conclude that the gender prediction algorithm performs well on this sample dataset. We can also treat the AUC measure as a general measure of predictiveness (Fawcett, 2003) and use it for comparison of different prediction methods.

Brier Score

Classification errors can be misleading metrics of gender prediction effectiveness, especially in the case when we have a great disproportion of female and male labels in the dataset. For example, in our `titles` dataset we have 87% male titles. If we predict male for each item in the dataset, our classification error will be only 13% due to small true proportion of female items.

```
R> titlesCoded <- titles[titles[['genderCoded']] %in% c('female', 'male'), ]
R> cbind('N' = table(titlesCoded[['genderCoded']]),
+        '%' = round(prop.table(table(titlesCoded[['genderCoded']])) * 100, 0))
```

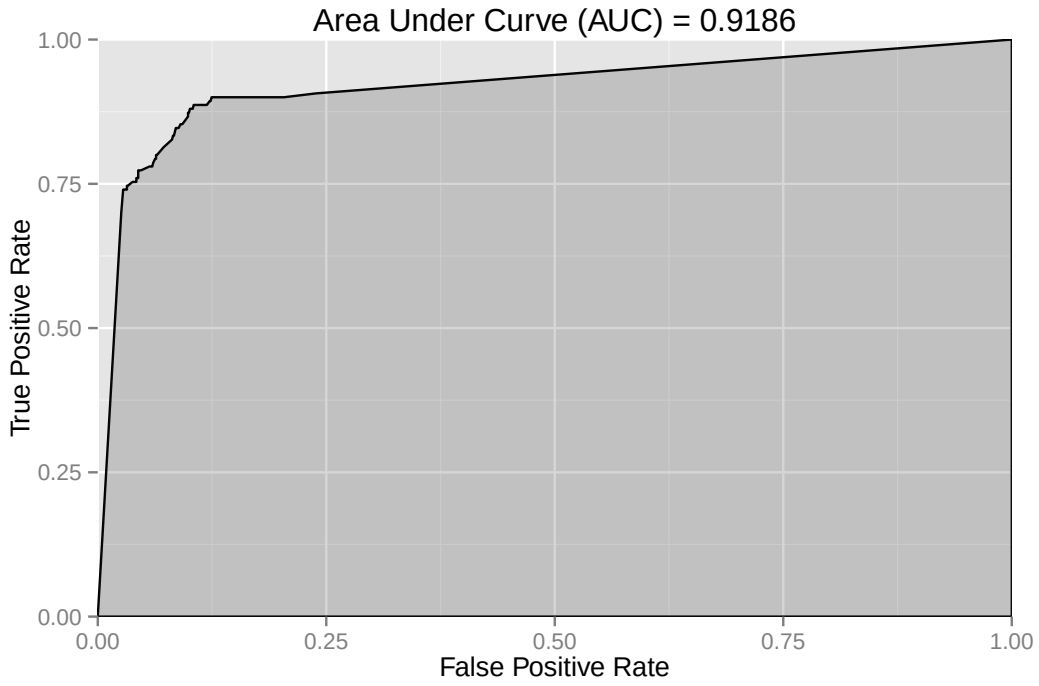

Figure 3: ROC curve for predicting female gender for the `titles` dataset. Large AUC suggests good prediction accuracy.

```
        N  %
female 151 13
male   994 87
R> indicators <-
+    classificationErrors(labels = titlesCoded[['genderCoded']],
+                         predictions = rep('male',
+                                           NROW(titlesCoded[['genderCoded']])))
R> unlist(indicators['errorCoded'])

errorCoded
 0.1318777
```

Predicting the most frequent class on every occasion in order to minimise the classification error rate is not an appropriate scoring rule and the prediction efficiency metrics should be penalised for such practices. That is why we use the *Brier Score*, which originated from a verification system of weather forecasts (Brier, 1950) and is known as a more proper classification accuracy score for predictions based on probabilities.

The *Brier Score* was defined as the sum of squared differences between `labels` values (ex. 1 if an item is a female and 0 if it is not) and `pred` values (probability if an item is a female) divided by the number of all items. The R package **verification** (NCAR – Research Applications Laboratory, 2014) offers an implementation of the *Brier Score* in the `brier` function.

```
R> library('verification')
R> brier(obs = d[['labels']], pred =  d[['pred']])[['bs']]

[1] 0.06577986
```

If an item's gender is correctly predicted with probability equal to 1, then the *Brier Score* is 0, and this is the best possible score. The worst score is 1.

We can also calculate presented and other metrics using a simulated bogus prediction algorithm. Such algorithm predicts the majority class (male labels) for all items. Both AUC and *Brier Score* are penalised since the bogus algorithm has just been guessing the gender using the most frequent class in this sample dataset.

Characteristics of prediction methods and prediction efficiency indicators	West et al. (2013)	Larivière et al. (2013)	**genderizeR** package
Source of first names data	US SSA	US Census	genderize.io
Extracted full first names	Yes	Yes	No
Gender probability threshold	0.95	0.91	0.50
Names count threshold	5	7200	1
Classification error rates (%):			
coded error rate	32.49	36.84	7.09
coded error rate without NA values	1.67	2.13	3.79
net gender bias error	1.67	2.13	1.42
Accuracy scores:			
AUC	0.926	0.920	0.927
Brier Score	0.099	0.109	0.097
Unpredicted gender (%):			
of all authorships	84.40	85.23	47.71
of authorships with full first names	31.56	35.22	4.65
of manually coded gender only	31.35	35.47	3.43

Table 3: Comparison of the effectiveness of gender prediction methods.

```
R> pred <- prediction(labels = d[['labels']],
+                     predictions = rep(0, NROW(d[['labels']])))
R> unlist(performance(pred, measure = 'auc')@y.values)

[1] 0.5
R> brier(obs = d[['labels']],
+        pred = rep(0, NROW(d[['labels']])))[['bs']]

[1] 0.12119
```

The comparison of methods

To compare different approaches of gender prediction, we need to reproduce those methods on the same sample that is drawn from the population of items with similar characteristics. We use an `authorships` dataset as such a sample, since predicting the gender of authors of the articles was the goal of two previously described studies.

Because the US SSA and the US Census data is available in the R packages, we can easily reproduce methods based on such data with parameters that were chosen by the authors of the studies. However, in the case of the method described by Larivière et al. (2013a), we can only reproduce gender prediction utilising the US Census data. Scraping and parsing other webpages or manually coding gender are highly resource intensive and our goal is to find a resource effective method. Nevertheless, we can later reproduce a confusion matrix from the study and try to estimate the real error for this mixed method.

The efficiency indicators of gender prediction for all three methods are presented in Table 3. For the third proposed method, we had not extracted first names from the authors' full names, and we let the `findGivenNames` function try to do this automatically.

As shown in Table 3, our proposed **method based on *genderize.io* API outperforms methods based on the US SSA and US Census data with the parameters set by the authors of these studies**. The percentage of items with unpredicted gender and *Brier Score* are clearly the lowest among all analysed methods.

It should be noted that using data sources other than the US Census with manual coding as done by Larivière et al. (2013a) would probably improve the proportion of items with predicted gender in our sample. However, it is difficult to assess the effectiveness of such a resource-consuming mixed method without comparable prediction metrics. The authors did not explicitly present the confusion matrix from their validation study, but we can recreate the matrix using materials from the analysed

	female	male	unisex	unknown	initials
female	146	4	2	24	48
male	22	267	9	51	156
unknown	35	178	10	9	39

Table 4: Recreated confusion matrix from Lariviere et al.'s (2013b) study with manually coded values (in rows) and automatically assigned values (in columns). The columns unknown, initials, and unisex from the recreated confusion matrix could be merged to one unknown column since the gender is unpredicted for these items.

studies, such as Table S3 *Number and percentage of distinct papers and of author-papers assigned a gender* and Table S6 *Percent male and female in each category* from the original paper (Larivière et al., 2013a, p. 6). From those original tables, we know the proportions of authorships classified as *female*, *male*, *unisex*, *unknown* and *initials*. The authors randomly sampled five samples of 1000 authorships from each category and manually coded whether those items were *female*, *male*, or *unknown* (Larivière et al., 2013a). From those known proportions, we can recreate the confusion matrix that sums to 1000 for a better perception (see Table 4).

Based on the recalculated data, the prediction indicators from the recreated confusion matrix are even worse than those in our comparative study where only the US Census data were used. The *coded error rate* is high and equals **43.3%**, since the proportion of manually coded items with unpredicted gender is also quite high **39.8%**. Moreover, the *gender bias error rate* in the original study is worse than in our comparative study based only on the US Census data, which suggests that **using additional data sources, manual coding, or unisex category will not necessarily increase the proportion of items with predicted gender and can also contribute to the bias of gender proportion estimates**.

Discussion

We cannot be sure how our method would perform on the same large datasets as in the described studies. Those studies were based on many non-English names and the *genderize.io* database has started gathering data mainly from public social profiles from the US and English-speaking countries, although it is still growing every minute and incorporating new data from other countries and languages.

The drawback of this data source is that people can put many different terms in their social network profiles that are not their first names. This could generate noise in the *genderize.io* database that could disturb gender prediction and introduce some bias if there are many instances of the same non-first-name terms used. Surprisingly, such crowd-sourced data work even better in our comparison study than very reliable and official data sources. In addition, we always can use a subset of the database with most reliable records.

We could also search for first names of papers' authors by taking into account the language in which they published their papers. Such an approach could be misleading, as many scientists publish papers in English, although it is not their native language. Another approach is to consider the proportion of males or females for a particular first name, while considering a year of birth of the person in question (Mullen, 2014). This historical method could improve gender predictions, but such birth data are often not available, especially in the discussed bibliometrics studies. However, if a year of birth is available, we could combine the two sources and use US Census data for years of birth from 1789 to 1930 and SSA data for newer ones as proposed by Mullen (2014). The particular problem with historic data is that the gender associated with a given name can change over time and the association can also differ geographically (Blevins and Mullen, 2015). The functions in the **genderizeR** package assume that we are looking for gender prediction based on global and contemporary gender-name associations.

The code in the **genderizeR** package works fast enough for many research purposes on datasets with more than 200 000 records of articles and their authors. Checking 108 023 unique terms through the *genderize.io* API took 45 minutes (2395.2 terms per minute). I have not tested and optimised the code for really large datasets yet, and by large datasets I mean millions of papers or authorships. However, some functions in the package are already implemented in parallel versions and can be used on larger datasets. I have already been experimenting with more efficient solutions from several other packages like **data.table** (Dowle et al., 2014), **dplyr** (Wickham and Francois, 2014), **stringr** (Wickham, 2012), **tm** (Feinerer and Hornik, 2014), and others. I am aware that many improvements could be made to the package and I am open for suggestions or contributions to the *GitHub* repository (`https://github.com/kalimu/genderizeR`).

The interesting prospect that can be explored in the future is the *genderizing* of some larger text corpuses. As we automatically try to ascribe gender to a title of a biographical article, we can additionally try to find first names in a larger text corpus and count how many references there are to females and males. It could be helpful, for example, in literature or media analysis.

Conclusion

Gender prediction is not as simple as it sometimes seems to be. In our data the gender category is not explicitly available for the researcher as it often is from, for example, a traditional survey question. When the name of a person in question is known, we can try to predict her or his gender based on the first name, even if we do not know which part of the record is a first name. This has many potential applications from big data analysis of authors of scientific papers to commercial applications where we can customise a commercial offer based on someone's gender without directly asking him or her about that.

The interest in gender identification seems to be increasing as we can see it in the studies done in recent years and in new IT tools that have emerged recently, such as *genderize.io*, R packages or code extensions for Ruby or Python that utilise gender data sources like *genderize.io* (Strømgren, 2015a).

New tools and a variety of open data sources (from the US Census, US SSA, and others) expand the possibilities of gender analysis, especially when they are in easy-readable machine formats. However, there is a shortage of proper validation studies that could show how effective the methods are that have been used for gender predictions. There is also a need to more explicitly present report prediction efficiency metrics that can be comparable with other methods across different studies.

From comparison of the three methods in this study, we can now assume that using *genderize.io* API as a gender data source is probably the best currently available approach. However, the outcomes could vary in different contexts, so it is always recommended to perform a validation study of different methods and compare a set of predictions metrics in order to select the most effective approach. Moreover, one should be cautious when using combined data sources or human coders as that can also have an adverse effect on some indicators of gender prediction efficiency.

Acknowledgements

This work is a result of many discussions with Marcin Kozak and Olesia Iefremova about different problems in bibliometrics studies, some of which the **genderizeR** package is intended to resolve. It would not exist without encouragement from Professor Kozak and his challenging questions (for which I am really very thankful). I would also like to thank Olesia Iefremova for manually checking and coding genders for over one thousand titles of biographical articles (which is a quite tedious and time-consuming activity). Last but not least, Casper Strømgren should be given special acknowledgements for practical implementation of the simple yet powerful idea of using crowd-sourced, publicly available data for gender prediction.

Bibliography

Web of Science, 2014. URL http://apps.webofknowledge.com. [p22]

C. Blevins and L. Mullen. Jane, John ... Leslie? A historical method for algorithmic gender prediction. *Digital Humanities Quarterly*, 9, 2015. URL http://www.digitalhumanities.org/dhq/vol/9/3/000223/000223.html. [p17, 35]

G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1): 1–3, 1950. doi: 10.1175/1520-0493(1950)078. [p30, 33]

M. Dowle, T. Short, S. Lianoglou, and A. Srinivasan. *data.table: Extension of data.frame*, 2014. URL http://CRAN.R-project.org/package=data.table. R package version 1.9.2, with contributions from R. Saporta and E. Antonyan. [p35]

B. Efron. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983. doi: 10.2307/2288636. [p19, 30]

B. Efron. How biased is the apparent error rate of a prediction rule? *Journal of the American Statistical Association*, 81(394):461–470, 1986. doi: 10.2307/2289236. [p30]

B. Efron and R. Tibshirani. Improvements on cross-validation: The .632 bootstrap method. *Journal of the American Statistical Association*, 92(438), June 1997. [p30, 31]

T. Fawcett. ROC graphs: Notes and practical considerations for data mining researchers. Technical report, Intelligent Enterprise Technologies Laboratory. Hawlett-Packard Laboratories, Palo Alto, CA, USA, 2003. [p30, 31, 32]

I. Feinerer and K. Hornik. *tm: Text Mining Package*, 2014. URL http://CRAN.R-project.org/package=tm. R package version 0.6. [p21, 35]

I. Feinerer, K. Hornik, and D. Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25 (5):1–54, Mar. 2008. URL http://www.jstatsoft.org/v25/i05/. [p21]

W. Jiang and R. Simon. A comparison of bootstrap methods and an adjusted bootstrap approach for estimating the prediction error in microarray classification. *Statistics in Medicine*, 26(29):5320–5334, Dec. 2007. doi: 10.1002/sim.2968. [p31]

V. Larivière, C. Ni, Y. Gingras, B. Cronin, and C. R. Sugimoto. Supplementary information to "Bibliometrics: Global gender disparities in science" (Comment in Nature 504, 211–213; 2013). *Nature*, 504, 12 2013a. [p17, 18, 20, 34, 35]

V. Larivière, C. Ni, Y. Gingras, B. Cronin, and C. R. Sugimoto. Bibliometrics: Global gender disparities in science. *Nature*, 504:211–213, 12 2013b. doi: 10.1038/504211a. [p17]

L. Mullen. *gender: Predict Gender From Names Using Historical Data*, 2014. URL https://github.com/ropensci/gender. [p19, 35]

NCAR – Research Applications Laboratory. *verification: Weather Forecast Verification Utilities.*, 2014. URL http://CRAN.R-project.org/package=verification. R package version 1.41. [p33]

J. A. Ramey. *sortinghat*, 2013. URL http://CRAN.R-project.org/package=sortinghat. R package version 0.1. [p21]

T. W. Rinker. *qdap: Quantitative Discourse Analysis Package*. University at Buffalo/SUNY, Buffalo, New York, 2013. URL http://github.com/trinker/qdap. Version 2.2.1. [p18]

T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer. ROCR: Visualizing classifier performance in R. *Bioinformatics*, 21(20):3940–3941, 2005. URL http://rocr.bioinf.mpi-sb.mpg.de. [p31]

Social Security Administration. Official social security website, 2015. URL http://www.ssa.gov/oact/babynames/limits.html. [p19]

C. Strømgren. genderize.io, 2015a. URL http://genderize.io. [p19, 20, 36]

C. Strømgren. store.genderize.io, 2015b. URL https://store.genderize.io. [p19]

United States Census Bureau. Frequently occurring surnames from Census 1990 – Names files, 2015a. URL http://www.census.gov/topics/population/genealogy/data/1990_census/1990_census_namefiles.html. [p18]

United States Census Bureau. Frequently occurring surnames from the Census 2000, 2015b. URL http://www.census.gov/topics/population/genealogy/data/2000_surnames.html. [p18]

K. Wais. *genderizeR: Gender Prediction Based on First Names*, 2016. URL http://CRAN.R-project.org/package=genderizeR. R package version 2.0.0. [p20, 21]

J. D. West, J. Jacquet, M. M. King, S. J. Correll, and C. T. Bergstrom. The role of gender in scholarly authorship. *PLOS ONE*, 8, 7 2013. [p17, 19, 20]

H. Wickham. *stringr: Make It Easier To Work With Strings*, 2012. URL http://CRAN.R-project.org/package=stringr. R package version 0.6.2. [p21, 35]

H. Wickham. *babynames: US Baby Names 1880–2013*, 2014. URL http://github.com/hadley/babynames. R package version 0.1. [p19]

H. Wickham and R. Francois. *dplyr: A Grammar of Data Manipulation*, 2014. URL http://CRAN.R-project.org/package=dplyr. R package version 0.3.0.2. [p35]

Kamil Wais
University of Information Technology and Management in Rzeszow
Sucharskiego 2, 35-225 Rzeszow
Poland
kamil.wais@gmail.com