Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

FLAME: An R Package for a Fast Large-scale Almost Matching Exactly Approach to Causal Inference

Chia-Rui (Jerry) Chang
Duke University

Sudeepa Roy
Duke University

Cynthia Rudin
Duke University

Alexander Volfovsky
Duke University

Abstract

The Fast Large-scale Almost Matching Exactly (FLAME) algorithm performs matching of treatment and control units in the potential outcomes framework for large categorical datasets. FLAME creates matches that include as many covariates as possible, and sequentially drops covariates that are less useful based on a match quality measure (Match Quality – MQ). MQ combines two important elements – it considers predictive power from machine learning on a hold out training set, and a balancing factor to ensure that it does not remove a covariate that would ruin overlap between treatment and control groups. This work introduces the **FLAME** R package. We demonstrate the algorithm and its implementation on simulated data and real world data. We also show how to retrieve causal estimands such as average treatment effect (ATE) and conditional average treatment effect (CATE) for interpretation.

Keywords: Causal inference, database systems, machine learning, matching, R.

1. Introduction

In health or social science, causal inference goes beyond simple correlations or model-based predictions to evaluate the effect of a certain treatment or intervention. For observational data in particular, a natural and interpretable approach to estimate treatment effects with low bias is through the matching of treated and control units. Matching on covariates allows practitioners to assess the quality of the data and analysis more easily, leading to better

hypotheses about possible confounding, and allowing a high-quality granular justification for decisions based on the basis of the matched groups.

However, most current matching methods either do not match directly on covariates, or are not high quality because they use an inflexible predefined distance metric that causes matched groups to be defined suboptimally. These methods thus cannot be used for conditional average treatment effect (CATE) estimation, and thus may not serve some of the main interpretability benefits of matching, discussed above. For instance, propensity score matching, or other methods that project the data onto a subspace, do not usually produce matches that are meaningful in the original covariate space. These methods often match observations together that have wildly different covariates from each other. Other matching methods that use a fixed distance metric, such as nearest neighbor matching or coarsened exact matching (Iacus, King, and Porro 2012), tend to lose accuracy in high dimensions because the distance metric ceases to be useful; these methods also have problems with irrelevant covariates, because the irrelevant covariates start to dominate the distance metric between observations. In other words, because the distance metric is manually defined, there is no guarantee that the quality of matching results is any good.

The FLAME algorithm (Roy, Rudin, Volfovsky, and Wang 2017) aims to avoid these standard problems with matching discussed above. FLAME operates under the standard potential outcomes framework with categorical covariates and binary treatment. FLAME first uses as many covariates as possible to create almost-exact matches, and sequentially drops covariates that are less predictive of the outcomes. By doing this, FLAME retains the important information needed to construct high-quality almost-exact matches. FLAME does not depend on a fixed distance measure, avoiding potential model misspecification of the distances between units. Particularly, the distance measure of FLAME is data-adaptive, using machine learning on a hold-out training set for determining which covariates are safe to drop in order to retain high-quality matches. This way, the distance between observations depends on actual covariate values – there is no latent representation (or propensity score projection) of the covariates. Because of this, FLAME's matches are interpretable, and allow estimation of not only average treatment effect (ATE) but also conditional average treatment effect (CATE). Because FLAME matches directly on covariates, using as many covariates as it can, we say that its matches are "almost exact."

This work introduces the the **FLAME** R package. Within this package, FLAME has two scalable implementations: one that leverage bit-vector operations for data that can fit in main memory, and a database implementation for data sets that are too large to fit in memory, using PostgreSQL (The PostgreSQL Development Team 2018) and SQLite (The SQLite Development Team 2018). This database method can operate on millions of observations, larger than the scale that most other matching methods are capable of. The **FLAME** package evaluates the predictive quality of each covariate at each step of the algorithm using machine learning models from Python scikit-learn package (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot, and Duchesnay 2011) on the hold-out training set. It also considers covariate balance when determining which covariates to drop, so that the groups do not become too imbalanced.

The following paper is organized as follows. In Section 2, we summarize the FLAME algorithm of Roy *et al.* (2017). Section 3 specifies the requirements for package installation. In Section 4, we introduce two datasets – one simulated (permitting for ground truth evaluation) and the other from the U.S. Centers for Disease Control and Prevention, and demonstrate the

application of **FLAME** to both datasets in Section 5. Section 6 summarizes how to retrieve **FLAME** results for interpretation and granular analysis.

2. The FLAME algorithm

We present the assumptions of FLAME in section 2.1, and introduce the algorithm in section 2.2.

2.1. Assumptions

Consider a population table $D = [X, Y, T]$, where each row corresponds to a *unit* (also known as *individual* or *observation*). X is a n x p data matrix. $X_1, X_2, ..., X_p$ are covariates. In order for exact matches to occur, values of covariates must be either categorical or binned continuous data. T is the treatment assignment, which takes binary values 1 (treated) and 0 (control). Y is the outcome variable. Denote $Y(0)$ as the outcomes of units in the control group, and $Y(1)$ as the outcomes of units in the treatment group. We make Stable Unit Treatment Value Assumption (SUTVA) (Rubin 1980) and strong ignorability assumption (Rosenbaum and Rubin 1983). SUTVA specifies that the outcome of one unit is not affected by the treatment assignment of any other units. Strong ignorability (also known as "ignorability", "no hidden bias", or "unconfounded") assumes that (1) the treatment assignment (T) is conditionally independent from the potential outcomes $(Y(0), Y(1))$ given covariates (X): $T \perp (Y(0), Y(1))|X$, and that (2) the probability of a unit assigned to treatment group is positive: $0 < P(T = 1|X) < 1$ for all X.

2.2. The generic FLAME algorithm

Algorithm 1 demonstrates the generic FLAME algorithm. Denote $D = [X, Y, T]$ as the input data, $D^H = [X^H, Y^H, T^H]$ as the holdout training set, and covariates as $J = \{j_1, j_2, ..., j_p\}$. X is the covariate (an $n \times p$ data matrix), Y is the outcome (an $n \times 1$ vector), and T is the treatment indicator (an $n \times 1$ vector). At each iteration l, FLAME finds matches with unmatched data D_l^{um} ($D_l^{um} \subseteq D$) using covariate subset J_l ($J_l \subseteq J$). The results include matched data D_l^m and matched groups MG_l.

At **Step 1 - 2**, initialize $l = 0$ and call function `BasicExactMatch` to find exact matches in the complete dataset $D = D_0 = [X, Y, T]$ using *all* covariates $J = J_0 = \{j_1, j_2, ..., j_p\}$. Each matched group $mg \in MG_0$ should include at least one treatment unit and one control unit.

The **while loop** at **Step 3** is the stopping condition. If there is no more covariate to eliminate ($J_l = \emptyset$), the algorithm stops. We also stop FLAME if the unmatched data contains no control unit ($D_{l,T=0}^{um} = \emptyset$) or no treatment unit ($D_{l,T=1}^{um} = \emptyset$). At **Step 4 -5**, increment l and remove matches found in the previous iteration. In order to find new matches, we have to reduce the dimension of the data. Thus, we choose covariate j_l^* from J_{l-1} to remove such that removing it maximizes Match Quality (MQ). MQ consists of two components - Balancing Factor (`BF`) and Predictive Error (`PE`), computed through -`PE` + tradeoff $\times$ `BF`. `BF` assesses the proportion of units that can be matched. `PE` evaluates the predictive quality of covariate subsets, built upon machine learning models such as ridge regression and decision tree. In short, MQ is a bias-variance tradeoff between producing more matches and having better prediction.

At **Step 6 - 7**, we update new covariate subset ($J_l = J_{l-1} \backslash j_l^*$) and find new matched groups

(MG_l) and matched data (D_l^m). We continue this process until any of the stopping condition is met. Once the algorithm stops, it returns a set of covariate subset, matched groups, and matched data as the final output.

Algorithm 1 The generic FLAME algorithm

Input:

(i) Input data $D = [X, Y, T]$

(ii) Holdout training set $D^H = [X^H, Y^H, T^H]$

Output:

(i) A set of covariates used for matching $\{J_l\}_{l\geq 0}$

(ii) A set of matched groups $\{MG_l\}_{l\geq 0}$

(iii) A set of matched data $\{D_l^m\}_{l\geq 0}$

Initialize $l = 0$, $D_l^{um} = [X, Y, T]$, $J_l = \{j_1, j_2, ..., j_p\}$, $MG = \emptyset$
$(D_l^m, MG_l) =$ `BasicExactMatch`(D_l^{um}, J_l) (find matches using all covariates)
while (i) $J_l \neq \emptyset$ and (ii) $D_{l,T=1}^{um} \neq \emptyset$ and (iii) $D_{l,T=0}^{um} \neq \emptyset$ **do**
    $l = l + 1$
    $D_l^{um} = D_{l-1}^{um} \backslash D_{l-1}^m$ (remove matches)
    $J_l = J_{l-1} \backslash j_l^*$ (new covariate subset with feature(s) removed)
    $(D_l^m, MG_l) =$ `BasicExactMatch`(D_l^{um}, J_l) (find matches using new covariate subset)
return $\{J_l, MG_l, D_l^m\}_{l\geq 0}$ (return covariate subset, matched groups, and matched data at each iteration)

Algorithm 2 presents the `BasicExactMatch` function. The inputs to `BasicExactMatch` include (1) unmatched data D^{um} and (2) covariate subset J_l. At **Step 1**, the algorithm performs exact matching on J_l, where it finds units with the same covariate values. Since each matched group should contain at least one treated unit and one control unit, we remove group not satisfying such constraint at **Step 2**. After matched groups are formed, we retrieve matched data D^m from D^{um} at **Step 3**. Lastly, D^m and M are returned. The implementation of `BasicExactMatch` uses bit vectors and SQL query.

Algorithm 2 BasicExactMatch

Input:

(i) Unmatched data $D^{um} = (X, Y, T)$

(ii) Covariate subset $J_l \subseteq J$

Output:

(i) Matched data D^m

(ii) Matched groups MG

$M_{raw} =$ `group-by`(D^{um}, J_l) (Perform exact matching on J_s and form group for units with the same covariate values)
$M =$ `prune`(M_{raw}) (remove groups that do not satisfy the matching constraint)
$D^m =$ Get subset of D^{um} from units in M
return $\{D^m, M\}$

3. Package requirements and installation

The **FLAME** package can be installed from https://github.com/chiarui424/FLAME. To install the pakcage, type the following command in R.

```
R> devtools::install_github("chiarui424/FLAME")
R> library(FLAME)
```

The package requires input data to have specific format. First, input data should be a R *Data Frame*. Second, all covariates in the input data should be categorical covariates, represented by *factor* data type. If there are continuous covariates, consider regrouping or binning. Third, input data should contain (1) covariates in *categorical* data type, (2) outcome variable in *numeric* data type, and (3) variable specifying a unit is treated or control (treated = 1, control = 0) in *factor* data type. Lastly, though there are no requirements for input data column names, the column order should follow *[covariates, outcome, treated]*. Table 1 is an example of input data with n units and p covariates. Holdout training set should also follow the same format.

	x_1	x_2	...	x_{p-1}	x_p	outcome	treated
R data type	factor	factor	factor	factor	factor	numeric	factor
unit 1	0	1	...	1	2	3.8	1
unit 2	1	0	...	1	0	1.36	0
unit 3	0	1	...	0	1	-7.25	0
...	...	...	...	...	...	...	...
unit n	0	1	...	1	0	20	1

Table 1: Input data example

Since **FLAME** relies on python and database management systems, other requirements to run the package include:

- **FLAME** requires installation of python (Python Software Foundation 2018), specifically with at least python 2.7 version. If your computer system does not have python 2.7, install from https://www.python.org/downloads/.
- For database systems implementation, **FLAME** package provides two versions - SQLite and PostgreSQL. PostgreSQL requires installation of external database system but it is faster. SQLite does not require external database system but is slower. If your computer does not have PostgreSQL installed, install it from https://www.postgresql.org/download/. For connecting and setup of PostgreSQL server, please refer to the tutorial at http://www.postgresqltutorial.com/connect-to-postgresql-database/.
- It is required to name the connection as **db** once connected to the database. We will demonstrate how to implement it in section 5.

Table 2 list all functions in **FLAME**.

4. Data

Function	Description
`ATE`	Compute average treatment effect (ATE) of all mathed groups
`AVG_EFFECT`	Compute the treatment effects of matched group(s)
`CATE`	Get the size and CATE of matched group(s)
`CATE_plot`	Summarize CATEs of all matched groups by Boxplot
`Data_Generation`	Generate simulated data
`FLAME_bit`	Bit vectors implementation
`FLAME_PostgreSQL`	PostgreSQL database implementation
`FLAME_SQLite`	SQLite database implementation
`MATCH`	Get units in matched group(s) given certain covariate combination
`summary_plot`	Visualize matching process

Table 2: Functions in the **FLAME** package

We introduce two datasets for package demonstration, one from simulation and the other from Centers for Disease Control and Prevention. For simulated data, function `Data_Generation` generates the data with specific characteristics and known treatment effects.

$$y_i = \sum_{j=1}^{n} \alpha_j x_j + T_i \sum_{j=1}^{n} \beta_j x_j + T_i \cdot U \sum_{j=1..5,\gamma=1...5,\gamma>j} x_j x_\gamma \tag{1}$$

$T_i \in \{0, 1\}$ indicates whether a unit i is treated or control, and U represents the coefficient of non-linear term. Here, $x_j \sim$ Bernoulli(0.5), $\alpha_j \sim N(10s, 1)$ with $s \sim Uniform\{-1, 1\}$, and $\beta_j \sim N(1.5, 0.15)$. For package demonstration, the simulated data contains 5,000 treated units and 5,000 control units with $U = 5$. In addition, there are 10 important covariates and 5 unimportant covariates. Important covariates are those used to generate the true outcome y_i, whereas unimportant covariates are random samples from Bernoulli(0.5). For the simulated data below, $x_1, x_2, ..., x_{10}$ are important covariates and $x_{11}, x_{12}, ..., x_{15}$ are unimportant covariates. We assume holdout training set to be the same as simulated data. The following code is used to generate simulated data.

```
R> set.seed(1234) # set seed for reproducibility
R> sim_data = FLAME::Data_Generation(num_control = 5000, num_treated = 5000,
+                                    num_cov_dense = 10, num_cov_unimportant = 5, U = 5)
R> sim_holdout = sim_data
R> head(sim_data)

  x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 outcome treated
1  0  0  0  0  0  0  1  0  0   1   1   1   0   0   1 -11.541       0
2  1  0  0  0  1  0  1  1  0   1   1   0   0   1   1   0.547       0
3  1  1  1  0  1  0  0  0  0   1   1   1   0   0   1  -2.263       0
4  1  1  1  0  1  0  0  1  0   0   1   1   0   1   1   6.491       0
5  1  0  0  0  1  1  1  1  0   1   1   0   1   1   0  -7.093       0
6  1  1  0  0  1  0  1  0  1   0   1   0   1   0   0   0.454       0
```

For real world data, we use U.S. 2010 natality dataset (Centers for Disease Control and Prevention 2010). We randomly sample 10 percent of the data, where there are 219,041

units with 86 covariates. Each unit is a pregnant mother. Covariates are mother's health conditions during pregnancy. The outcome indicates whether the infant has abnormal health conditions after birth. Abnormal conditions include *assisted aentilation, admission to NICU, surfactant therapy received, antibiotics received for suspected neonatal sepsis and seizures or serious neurologic dysfunction.* Infant with abnormal conditions has outcome '1', while infant with normal conditions has outcome '0'. Since we would like to evaluate the effect of smoking during pregnancy on infant abnormal health conditions, mothers who smoke cigarettes during pregnancy are assigned to be treated. There are 20,551 treated units and 198,490 control units.

5. Running the FLAME package

We demonstrate running the package on simulated data in section 5.1 and natality data in section 5.2. The main functions for running FLAME are `FLAME_bit`, `FLAME_SQLite` and `FLAME_PostgreSQL`. The inputs for all three functions include input data and holdout training set, where the format of the data should follow the requirements in section 3.

Note that there are other optional inputs related to Match Quality. If these optional inputs are not specified, the default is set to Ridge Regression with tradeoff parameter of 0.1. We focus on running major functions in this section, so Match Quality is set to default. The details of specifying Match Quality parameters is introduced in section 7.

5.1. Application to simulated data

`FLAME_bit`: We apply `FLAME_bit` function to the simulated data and save the result as `result_bit`. Note that if we want to compute the variance of each matched group, we can specify optioanl argument `compute_var = TRUE`. The variance of each matched group is the sum of the variance of the treated units and the variance of the control units, so each matched group should contain at least 2 treated units and 2 control units. Therefore, it is expected that less matched units will be found in earlier iteration if `compute_var = TRUE`.

```
R> result_bit <- FLAME_bit(data = sim_data, holdout = sim_holdout)
```

`FLAME_PostgreSQL`: Database system implementation with PostgreSQL version is developed with package **RPostgreSQL** (Conway, Eddelbuettel, Nishiyama, Prayaga, and Tiffin 2017). It requires additional input `db`, name of the connection. Note that it is required to name the connection as **db** as mentioned in section 3. The following code connects to PostgreSQL in R, runs `FLAME_PostgreSQL` function, saves the result as `result_PostgreSQL`, and disconnects **db** once done.

```
R> drv = dbDriver('PostgreSQL')
R> db = dbConnect(drv, user="postgres", dbname="FLAME", host='localhost',
+                 port=5432, password = 'new_password')
R> result_PostgreSQL = FLAME_PostgreSQL(db = db, data = sim_data, holdout = sim_holdout)
R> dbDisconnect(db)
```

`FLAME_SQLite`: Database system implementation with SQLite version is developed with package **RSQLite** (MÃijller, Wickham, James, and Falcon 2018). Similar to `FLAME_PostgreSQL`,

`FLAME_SQLite` also requires additional input `db`, where the connection must be named as **db**. The following code connects to a temporary database in R, runs `FLAME_SQLite` function, saves the result as `result_SQLite`, and disconnects **db** once done. Note that the name of the temporary database can be arbitrary. Here, we name it as `"tempdb"`.

```
R> db = dbConnect(SQLite(),"tempdb")
R> result_SQLite = FLAME_SQLite(db = db, data = sim_data, holdout = sim_holdout)
R> dbDisconnect(db)
```

FLAME returns the same outputs regardless of bit vector or database implementations. The outputs include:

1. `covariate_list`: set of covariates FLAME performs matching at each iteration ($\{J_l\}_{l\geq 0}$ in algorithm 1)
2. `matched_group`: set of matched groups ($\{MG_l\}_{l\geq 0}$ in algorithm 1)
3. `match_quality`: Match Quality at each iteration
4. `matched_data`: The original data with additional column **matched**, indicating the number of covariates FLAME performs matching. For example, if a unit is matched with 5 covariates, then the **matched** column of this unit is 5. If a unit is never matched, then **matched** will be 0.

We compare the results of all three implementations, which yield the same results.

- We apply summary function to examine average treatment effect and number of matches.

```
R> summary(result_bit) #bit vectors

[1] "Number of units matched =  9950"
[1] "Average treatment effect =  19.6186217266224"

R> summary(result_SQLite) #PostgreSQL

[1] "Number of units matched =  9950"
[1] "Average treatment effect =  19.6186217266224"

R> summary(result_PostgreSQL) #SQLite

[1] "Number of units matched =  9950"
[1] "Average treatment effect =  19.6186217266224"
```

- The covariates FLAME performs matching at the 8^{th} iteration.

```
R> result_bit$covariate_list[[8]]  #bit vectors

[1] "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x10"
```

```
R> result_PostgreSQL$covariate_list[[8]] #PostgreSQL

[1] "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x10"

R> result_SQLite$covariate_list[[8]] #SQLite

[1] "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x10"
```

- All matched groups at the 8^{th} iteration. Column **effect** is the conditional average treatment effect (CATE) and column **size** is the number of units in each matched group.

```
R> head(result_bit$matched_group[[8]]) #bit vectors

  x1 x2 x3 x4 x5 x6 x7 x10  effect size
1  0  0  0  0  0  1  0   0 -0.2217    9
2  0  0  0  1  0  0  1   0  1.6640   11
3  0  0  0  0  1  0  1   0  1.8027    8
4  0  1  0  0  0  0  0   0  1.8442   10
5  0  0  0  0  0  0  1   1  3.1416    6
6  0  0  0  0  0  0  1   0  4.4867    8

R> head(result_PostgreSQL$matched_group[[8]]) #PostgreSQL

  x1 x2 x3 x4 x5 x6 x7 x10  effect size
1  0  0  0  0  0  1  0   0 -0.2217    9
2  0  0  0  1  0  0  1   0  1.6640   11
3  0  0  0  0  1  0  1   0  1.8027    8
4  0  1  0  0  0  0  0   0  1.8442   10
5  0  0  0  0  0  0  1   1  3.1416    6
6  0  0  0  0  0  0  1   0  4.4867    8

R> head(result_SQLite$matched_group[[8]]) #SQLite

  x1 x2 x3 x4 x5 x6 x7 x10  effect size
1  0  0  0  0  0  1  0   0 -0.2217    9
2  0  0  0  1  0  0  1   0  1.6640   11
3  0  0  0  0  1  0  1   0  1.8027    8
4  0  1  0  0  0  0  0   0  1.8442   10
5  0  0  0  0  0  0  1   1  3.1416    6
6  0  0  0  0  0  0  1   0  4.4867    8
```

5.2. Package illustration with U.S. Natality Data

In this section, we demonstrate how to apply **FLAME** to 2010 U.S. Natality Data. To recap, the data aims to study the effect of smoking during pregnancy on the health outcome of infants. Each unit is a pregnant mother. Covariates indicate mothers' health condition

during pregnancy. Treatment group are mothers who smoke during pregnancy. Outcome indicates whether the health conditions of infant are normal ('0') or abnormal ('1'). There are 20,551 treated units, 198,490 control units, and 86 covariates. Holdout training set is the same as input data. Given the scale of the data, we apply `FLAME_PostgreSQL` for faster implementation.

First, we connect to the database in R and name the connection as **db**. Second, we run `FLAME_PostgreSQL`. Here, ridge regression with tradeoff parameter of 0.1 is used to compute Match Quality (default). Third, we disconnect **db** once the matching is done.

```
R> drv <- dbDriver('PostgreSQL')
R> db<- dbConnect(drv, dbname="FLAME", host='localhost',
+                 port=5432, user="postgres", password = 'new_password')
R> result_natality <- FLAME_PostgreSQL(db = db, data, holdout)
R> dbDisconnect(db)
```

The average treatment effect (ATE) of the population is approximately 0.04834. This indiates smoking during pregnancy increases the chance of abnormal pregnancy outcomes by 4.8 %.

```
R> summary(result_natality)

[1] "Number of units matched =  153903"
[1] "Average treatment effect =  0.048349247835241"
```

6. Access detailed information for interpretation

After running FLAME, we can access detailed information such as conditional average treatment effect (CATE) of each matched group and average treatment effect (ATE) of the matched samples with functions `CATE`, `MATCH`, `AVG_EFFECT`, and `CATE_plot`. Table 3 summarizes the input parameters of each function.

Here, we demonstrate how to retrieve conditional average treatment effect (CATE) using results from section 5.

1. Let's look at the iteration when the FLAME algorithm performs matching with 8 covariates. These 8 covariates are:

   ```
   [1] "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x10"
   ```

   If we want to access all matched groups and each group's CATE at this iteration, apply `CATE` function by specifying parameter `FLAME_object` and `num_covs`.

   ```
   R> head(CATE(FLAME_object = result_bit, num_covs  = 8))

     x1 x2 x3 x4 x5 x6 x7 x10  effect size
   1  0  0  0  0  0  1  0   0 -0.2217    9
   2  0  0  0  1  0  0  1   0  1.6640   11
   ```

function	parameter	description
CATE	FLAME_object	object returned by applying the FLAME matching algorithm (result_bit, result_PostgreSQL, and result_SQLite)
	num_covs	number of covariates FLAME performs matching with
	cov_name	a *vector* of covariate names
	cov_val	a *vector* of covariate values, where the value position should match cov_name position (each element has to be *character* data type)
MATCH	FLAME_object	object returned by applying the FLAME matching algorithm (result_bit, result_PostgreSQL, and result_SQLite)
	cov_name	a *vector* of covariate names
	cov_val	a *vector* of covariate values, where the value position should match cov_name position (each element has to be *character* data type)
AVG_EFFECT	CATE_object	object returned by applying CATE function
CATE_plot	CATE_object	object returned by applying CATE function

Table 3: Input parameters

```
3  0  0  0  0  1  0  1   0  1.8027    8
4  0  1  0  0  0  0  0   0  1.8442   10
5  0  0  0  0  0  0  1   1  3.1416    6
6  0  0  0  0  0  0  1   0  4.4867    8
```

2. Let's look at one of the matched groups. This matched group has the following covariate combination.

```
 x1  x2  x3  x4  x5  x6  x7 x10
"0" "0" "0" "0" "0" "1" "0" "0"
```

If we want to see the CATE and matched size of this covariate combination, we can apply CATE function by specifying all parameters.

```
R> cov_name  #covariate names

[1] "x1"  "x2"  "x3"  "x4"  "x5"  "x6"  "x7"  "x10"

R> cov_val #covariate values in character R data type

[1] "0" "0" "0" "0" "0" "1" "0" "0"

R> CATE(FLAME_object = result_bit, num_covs  = 8,
+        cov_name = cov_name, cov_val = cov_val)

  x1 x2 x3 x4 x5 x6 x7 x10  effect size
1  0  0  0  0  0  1  0   0 -0.2217    9
```

3. If we want to know which observations/units produce this covariate combination, we can apply MATCH function by specifying all parameters.

```
R> MATCH(FLAME_object = result_bit, cov_name = cov_name, cov_val = cov_val)

  x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 outcome treated
1  0  0  0  0  0  1  0  1  0   0   0   0   0   0   1  -7.307       0
2  0  0  0  0  0  1  0  0  0   0   1   1   0   1   1  -7.640       0
3  0  0  0  0  0  1  0  0  0   0   1   0   1   1   1  -7.640       0
4  0  0  0  0  0  1  0  1  0   0   0   1   1   1   0  -7.307       0
5  0  0  0  0  0  1  0  0  1   0   1   0   1   1   1  -7.695       1
6  0  0  0  0  0  1  0  0  1   0   1   0   0   0   0  -7.695       1
7  0  0  0  0  0  1  0  0  1   0   1   1   1   0   0  -7.695       1
8  0  0  0  0  0  1  0  0  1   0   0   1   0   0   0  -7.695       1
9  0  0  0  0  0  1  0  0  1   0   0   0   0   0   1  -7.695       1
  index
1  8741
2  8793
3  8803
4  8804
5  8929
6  8934
7  8954
8  8998
9  9080
```

4. Assume we want to see all matched groups with the covariate combination **x1 = 1, x3 = 0, x5 = 1, x7 = 0, x9 = 1**. Particularly, even when the FLAME algorithm performs matching with more than 5 covariates, we can still output all possible matches as long as it contains the specified covariate combination. We apply `CATE` function by specifying parameters `FLAME_object`, `cov_name`, and `cov_val`.

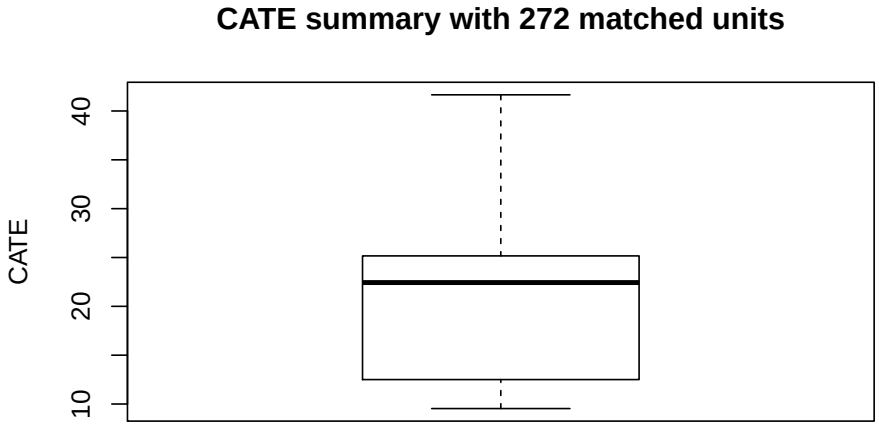
```
R> CATE_object <- CATE(FLAME_object = result_bit,
+                      cov_name = c("x1", "x3", "x5", "x7", "x9"),
+                      cov_val = c("1", "0", "1", "0", "1"))

[[1]]
    x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 effect size
168  1  0  0  0  1  0  0  0  1   0   0   1   1   0   1  9.534    2
231  1  0  0  0  1  0  0  1  1   0   1   0   0   0   0 11.055    2
232  1  0  0  0  1  0  0  1  1   0   1   1   0   1   0 11.055    2
233  1  0  0  0  1  0  0  1  1   0   0   0   1   1   0 11.055    2
234  1  0  0  0  1  0  0  1  1   0   1   1   0   1   1 11.055    2
289  1  0  0  0  1  1  0  0  1   1   0   1   0   0   0 12.350    2

[[2]]
    x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x13 x14 x15 effect size
137  1  0  0  0  1  0  0  0  1   1   1   0   1   0  10.90    3
142  1  0  0  0  1  1  0  0  1   0   1   0   0   1  10.98    2
146  1  0  0  0  1  0  0  1  1   0   0   0   0   0  11.05    2
```

```
193  1  0  0  0  1  1  0  1  1   0   1   1   1   1  12.50    2
281  1  0  0  1  1  0  0  0  1   1   1   0   1   1  22.20    2
297  1  0  0  1  1  0  0  1  1   0   1   1   0   1  22.35    2

[[3]]
    x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x13 x14 effect size
124  1  0  0  0  1  0  0  0  1   0   1   1   1  9.534    2
181  1  0  0  0  1  1  0  0  1   0   1   1   1 10.981    2
183  1  0  0  0  1  0  0  1  1   0   1   1   0 11.055    2
184  1  0  0  0  1  0  0  1  1   0   0   0   1 11.055    2
185  1  0  0  0  1  0  0  1  1   0   1   1   1 11.055    3
236  1  0  0  0  1  0  0  1  1   1   1   0   0 12.424    2

[[4]]
    x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x14 effect size
216  1  0  0  0  1  0  0  0  1   1   1   0  10.90    2
278  1  0  0  0  1  1  0  0  1   1   0   1  12.35    3
291  1  0  0  0  1  1  0  1  1   0   0   0  12.50    5
323  1  0  0  0  1  1  0  1  1   1   1   0  13.87    2
324  1  0  0  0  1  1  0  1  1   1   0   1  13.87    2
431  1  0  0  1  1  1  0  0  1   0   1   1  22.27    2

[[5]]
    x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 effect size
158  1  0  0  0  1  0  0  0  1   1   0  10.90    3
168  1  0  0  0  1  1  0  0  1   0   1  10.98    2
282  1  0  0  1  1  1  0  0  1   0   0  22.27    3
294  1  1  0  0  1  0  0  0  1   1   0  22.39    2
336  1  0  0  1  1  0  0  1  1   1   0  23.72    3
346  1  0  0  1  1  1  0  1  1   0   1  23.80    3

[[6]]
    x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 effect size
123  1  1  0  0  1  0  0  0  1   0  21.03    5
139  1  0  0  1  1  0  0  1  1   0  22.35    6
155  1  0  0  1  1  1  0  1  1   1  25.16    4
176  1  1  0  1  1  0  0  0  1   1  38.69    2
177  1  1  0  1  1  1  0  0  1   0  38.77   11
179  1  1  0  1  1  0  0  1  1   0  38.84    5

[[7]]
   x1 x2 x3 x4 x5 x6 x7 x9 x10 effect size
33  1  0  0  0  1  0  0  1   0  11.39    6
48  1  0  0  0  1  1  0  1   0  12.84    7
56  1  0  0  0  1  1  0  1   1  14.20    5
73  1  1  0  0  1  0  0  1   1  22.06    6
```

5. To compute estimated treatment effects of matched groups, we apply `AVG_EFFECT` function. Note that estimated treatment effect is the weighted average of CATEs, with weight being the number of units in each matched group. In addition, we can visualize the distribution of all CATEs in boxplot by applying `CATE_plot` function.

```
R> CATE_plot(CATE_object)
R> AVG_EFFECT(CATE_object)
```

```
[1] 23.49
```



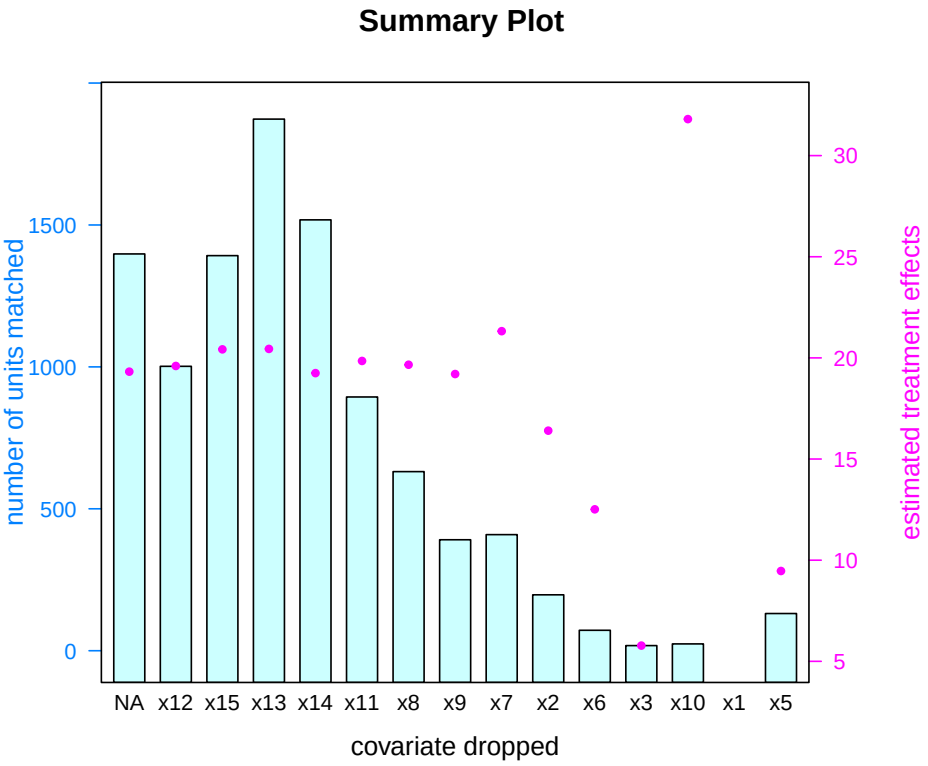
To get the summary of average treatment effect (ATE), we can apply function `ATE` and `summary_plot`. The input parameter of both functions is the object returned by applying the FLAME algorithm. In our example, it is `result_bit`, `result_PostgreSQL`, or `result_SQLite`. `ATE` computes average treatment effect for each given sub-population by a weighted average of the estimated treatment effects in each matched group. The weight is the number of units in each matched group.

```
R> FLAME::ATE(result_bit)
```

```
[1] 19.62
```

We can also apply `summary_plot` function to visualize the process of matching as the algorithm drops covariate iteratively. The x-axis is the covariate dropped at each iteration. The y-axis on the left is the number of units matched, and the y-axis on the right is the estimated treatment effects. Note that since all covariates are used for matching in the beginning, no covariate is dropped and is represented by NA in the covariate dropped axis.

```
R> FLAME::summary_plot(result_bit)
```



7. Specifying Match Quality

One of the key steps for FLAME is that it removes covariate iteratively according to Match Quality. To recap, MQ is computed as `PE` + tradeoff × `BF`. The implementation of the FLAME algorithm allows users to choose both tradeoff parameter and model to compute `PE`. First, users who prefer covariate subset that produces more matched units should choose higher tradeoff parameter, whereas users who prefer covariate subset that produces better predictive accuracy should choose lower tradeoff parameter. The default tradeoff parameter is set to 0.1. Second, users can choose appropriate statistical model to compute predictive error. The FLAME package currently provides four models; respectively they are Linear Regression, Ridge Regression, LASSO, and Decision Tree. The default model is set to ridge regression with L2 regularization parameter of 0.1. If users would like to use models not supported by the package, they can provide an input R function that computes predictive error. Table 4 summarizes the Match Quality parameters.

We demonstrate how to specify predictive models based on bit vectors implementation.

1. Linear Regression - specify this operation as `model = "Linear"`

```
R> FLAME_bit(data = data, holdout = holdout, model = "Linear")
```

2. Ridge Regression with L2 regularization parameter of 0.1 - specify this operation as `model = "Ridge", ridge_reg = 0.1`

```
R> FLAME_bit(data = data, holdout = holdout, model = "Ridge", ridge_reg = 0.1)
```

parameter	description
`tradeoff`	tradeoff parameter to compute Match Quality. Default = 0.1.
`PE_function`	statistical model specified in R function to compute predivtive error
`model`	options include "Linear" (linear regression), "Ridge" (ridge regression), "Lasso" Lasso, or "DecisionTree" (classification and regression tree). Default is set to "Ridge".
`ridge_reg`	L2 regularization parameter if `model = "Ridge"`. Default = 0.1.
`lasso_reg`	L1 regularization parameter if `model = "Lasso"`. Default = 0.1.
`tree_depth`	maximum depth of decision tree if `model = "DecisionTree"`.

Table 4: Optional parameters related to Match Quality

3. Lasso with L1 regularization regularization parameter of 0.1 - specify this operation as `model = "Lasso", lasso_reg = 0.1`

```
R> FLAME_bit(data = data, holdout = holdout, model = "Lasso", lasso_reg = 0.1)
```

4. Decision Tree with maximum depth of 8 - specify this operation as `model = "DecisionTree", tree_depth = 8`

```
R> FLAME_bit(data = data, holdout = holdout, model = "DecisionTree", tree_depth = 8)
```

If users would like to use the model not supported by the package, users can specify their own model with parameter `PE_function`. The inputs to `PE_function` consist of four parameters, summarized in table 5. Predictive error is the sum of mean squared error (MSE) for treated group and mean squared error (MSE) for control group. `outcome_treated` and `covs_treated` is used to compute the MSE for treated group, and `outcome_control` and `covs_control` is used to compute the MSE for control group.

`outcome_treated`	outcomes of treated units in *vector* data type
`outcome_control`	outcomes for control units in *vector* data type
`covs_treated`	covariate values for treated units in *matrix* data type
`covs_control`	covariate values for control units in *matrix* data type

Table 5: parameters of PE function

Here, we write a R function that computes predictive error based on support vector machine (SVM). The name of the function is `SVM_PE`, which is based on SVM model in **e1071** package (Meyer, Dimitriadou, Hornik, Weingessel, and Leisch 2017).

```
R> SVM_PE <- function(outcome_treated, outcome_control, covs_treated, covs_control) {
+
+    library(e1071) #load e1071 library
+
+    # MSE for treated group
+    model_svm <- svm(outcome_treated ~ covs_treated) # fit the data to SVM model
```

```
+    pred_treated <- predict(model_svm, covs_treated) #get fitted values
+    MSE_treated <- sum((outcome_treated - pred_treated)^2)/length(outcome_treated)
+
+    # MSE for control
+    model_svm <- svm(outcome_control ~ covs_control) # fit the data to SVM model
+    pred_control <- predict(model_svm, covs_control) #get predicted values
+    MSE_control <- sum((outcome_control - pred_control)^2)/length(outcome_control)
+
+    return(MSE_treated + MSE_control)
+  }
```

We then specify `PE_function = SVM_PE` and apply it to any of the FLAME implementation.

```
R> FLAME_bit(data = data, holdout = holdout, PE_function = SVM_PE)
```

Acknowledgments

We thank Tianyu Wang for providing Python code and technical assistance.

References

Centers for Disease Control and Prevention (2010). "Birth Data." Data retrieved from National Vital Statistics System, https://www.cdc.gov/nchs/nvss/births.htm.

Conway J, Eddelbuettel D, Nishiyama T, Prayaga SK, Tiffin N (2017). ***RPostgreSQL**: R Interface to the PostgreSQL Database System*. R package version 0.6-2, URL https://cran.r-project.org/web/packages/RPostgreSQL/index.html.

Iacus SM, King G, Porro G (2012). "Causal inference without balance checking: Coarsened exact matching." *Political analysis*, **20**(1), 1–24.

Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2017). ***e1071**: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. R package version 1.6-8, URL https://CRAN.R-project.org/package=e1071.

MÃijller K, Wickham H, James DA, Falcon S (2018). ***RPostgreSQL**: SQLite Interface for R*. R package version 2.1-1, URL https://cran.r-project.org/web/packages/RSQLite/index.html.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011). "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, **12**, 2825–2830.

Python Software Foundation (2018). *Python Software, Version 2.7*. URL http://www.python.org.

Rosenbaum PR, Rubin DB (1983). "The central role of the propensity score in observational studies for causal effects." *Biometrika*, **70**(1), 41–55.

Roy S, Rudin C, Volfovsky A, Wang T (2017). "FLAME: A Fast Large-scale Almost Matching Exactly Approach to Causal Inference." *arXiv preprint arXiv:1707.06315*.

Rubin DB (1980). "Bias reduction using Mahalanobis-metric matching." *Biometrics*, pp. 293–298.

The PostgreSQL Development Team (2018). *PostgreSQL*. URL https://www.postgresql.org.

The SQLite Development Team (2018). *SQLite*. URL https://www.sqlite.org.

Affiliation:

Chia-Rui (Jerry) Chang
Department of Statistical Science
Duke University
Durham, North Carolina
E-mail: chiarui424@gmail.com

Sudeepa Roy
Department of Computer Science
Duke University
Durham, North Carolina
E-mail: sudeepa@cs.duke.edu

Cynthia Rudin
Department of Computer Science and Electrical and Computer Engineering
Duke University
Durham, North Carolina
E-mail: cynthia@cs.duke.edu

Alexander Volfovsky
Department of Statistical Science
Duke University
Durham, North Carolina
E-mail: alexander.volfovsky@duke.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd