LightGBM's Road to CRAN

/jameslamb @_jameslamb

LightGBM is a light-weight Gradient Boosting Machine

A Communication-Efficient Parallel Algorithm for Decision Tree

Qi Meng[1,*], Guolin Ke[2,*], Taifeng Wang[2], Wei Chen[2], Qiwei Ye[2], Zhi-Ming Ma[3], Tie-Yan Liu[2]
[1]Peking University [2]Microsoft Research
[3]Chinese Academy of Mathematics and Systems Science
[1]qimeng13@pku.edu.cn; [2]{Guolin.Ke, taifengw, wche, qiwye, tie-yan.liu}@microsoft.com;
[3]mazm@amt.ac.cn

LightGBM: A Highly Efficient Gradient Boosting Decision Tree

Guolin Ke[1], Qi Meng[2], Thomas Finley[3], Taifeng Wang[1], Wei Chen[1], Weidong Ma[1], Qiwei Ye[1], Tie-Yan Liu[1]
[1]Microsoft Research [2]Peking University [3] Microsoft Redmond
[1]{guolin.ke, taifengw, wche, weima, qiwye, tie-yan.liu}@microsoft.com;
[2]qimeng13@pku.edu.cn; [3]tfinely@microsoft.com;

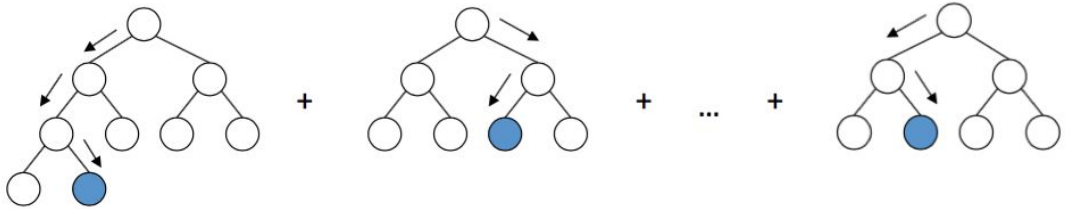
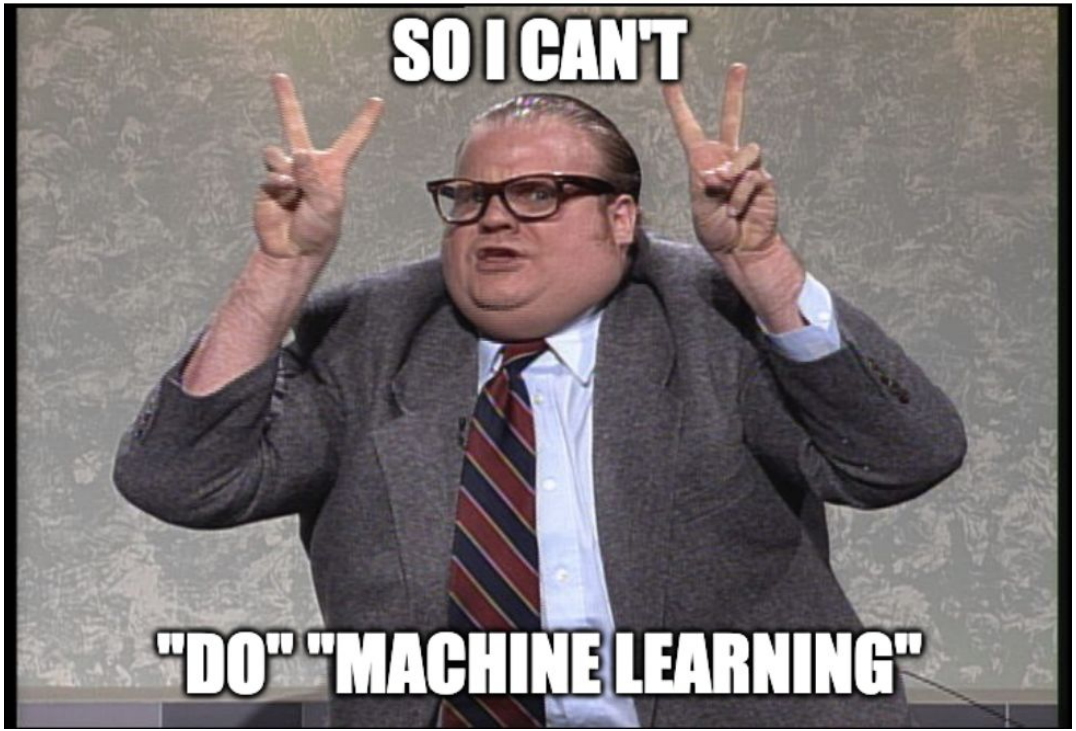


photo credit: "Gradient Boosting Explained", Alex Rogozhnikov

...and that is the extent of the machine learning you'll learn in this talk. I am not that good at machine learning and this is not about that.



LightGBM is implemented in C++, with many other extensions

```cpp
Boosting* Boosting::CreateBoosting(const std::string& type, const char* filename) {
  if (filename == nullptr || filename[0] == '\0') {
    if (type == std::string("gbdt")) {
      return new GBDT();
    } else if (type == std::string("dart")) {
      return new DART();
    } else if (type == std::string("goss")) {
      return new GOSS();
    } else if (type == std::string("rf")) {
      return new RF();
    } else {
      return nullptr;
    }
  } else {
    std::unique_ptr<Boosting> ret;
    if (GetBoostingTypeFromModelFile(filename) == std::string("tree")) {
      if (type == std::string("gbdt")) {
        ret.reset(new GBDT());
      } else if (type == std::string("dart")) {
        ret.reset(new DART());
      } else if (type == std::string("goss")) {
        ret.reset(new GOSS());
      } else if (type == std::string("rf")) {
        return new RF();
      } else {
        Log::Fatal("Unknown boosting type %s", type.c_str());
      }
      LoadFileToBoosting(ret.get(), filename);
    } else {
      Log::Fatal("Unknown model format or submodel type in model file %s", filename);
    }
    return ret.release();
  }
}
```

Maintained by us

- command-line interface
- Python package
- R package
- Java interface (via SWIG)
- GPU acceleration (via OpenCL)

Maintained by others

- languages → Julia, Go, Rust, .NET/C#
- frameworks → Spark, PMML, ONNX, treelite, Dask
- ...and many more!

From this point forward, “LightGBM” refers to the project, {lightgbm} refers to the R package. This talk is about the R package.

LightGBM has 9 maintainers.

{lightgbm} has 3.

- @guolinke **Guolin Ke** (C++ code / R-package / Python-package)
- @chivee **Qiwei Ye** (C++ code / Python-package)
- @btrotta **Belinda Trotta** (C++ code)
- @Laurae2 **Damien Soukhavong** (R-package)
- @jameslamb **James Lamb** (R-package)
- @wxchan **Wenxuan Chen** (Python-package)
- @henry0312 **Tsukasa Omoto** (Python-package)
- @StrikerRUS **Nikita Titov** (Python-package)
- @huanzhang12 **Huan Zhang** (GPU support)

I started contributing on LightGBM in March 2018. Between now and then, we've encountered a lot of issues needed to get the package ready for CRAN.

These are the main ones.

There are more!

- Misspelled parameter names in documentation
- Examples in documentation take more than 5 minutes to run
- Tests that write to disk leave files behind
- Mistakes compiler detection in install.libs.R using regular expressions
- Empty directories have to be preserved for install.libs.R to work
- Unnecessary dependencies in Suggests section of DESCRIPTION
- Tests use paths that don't work on Windows
- Tests fail because shQuote() produces different quotes on different OSes

- "Description" in DESCRIPTION file starts with package name
- Code called with "::" not in Imports
- Duplicate entries in man/ files
- Undocumented function arguments
- External library functions not referenced with "::"
- "Unused global variables" warning caused by {data.table} code
- @UseDynLib references 'lightgbm' but the .so/.dll is called 'lib_lightgbm'
- Make files have Windows-specific line endings
- C++ library calls functions that terminate sessions, like `abort` or `assert`
- C++ library prints with functions like `vprintf`, `sprintf`, etc. instead of directing output back to the R console
- C++ library does not call R_registerRoutines or R_useDynamicSymbols
- Installation code in install.libs.R requires that you be in the LightGBM repo
- .o / .a and other object files are included in package folder
- CMake is not part of the supported CRAN toolchain

CRAN is a repository for self-contained, properly-documented, portable, source-only, easily-installable R software packages. {lightgbm} had room for improvement in all of these areas.

self-contained

All package files bundled in a single .tar.gz. Can be installed without any assumptions about folder structure in the installation location.

properly-documented

All exported objects must have some minimal level of documentation.

portable

Works on many operating systems (e.g. Windows, Linux, Mac) and architectures (e.g. 32-bit, 64-bit)

source-only, easily installable

Executable files (.exe, .msi) or shared libraries (.dll, .so) cannot be bundled in the package.

R should be able to create these executables using just the code in the package and widely-available open source build tools

In theory, everything you need to know about building a CRAN-ready package is in “Writing R Extensions”.

Writing R Extensions

This is a guide to extending R, describing the process of creating R add-on packages, writing R documentation, R’s system and foreign language interfaces, and the R API.

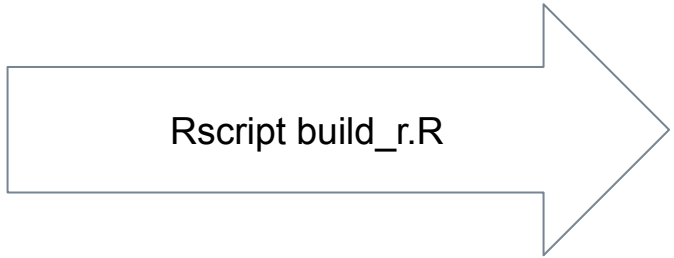
This manual is for R, version 4.0.0 (2020-04-24).

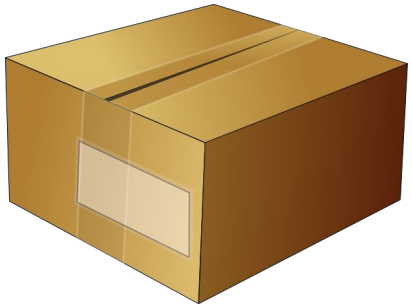
https://cran.r-project.org/doc/manuals/r-release/R-exts.html

[self-contained] All of an R package's contents have to be wrapped in one archive, that you can install anywhere.

[self-contained] {lightgbm} has to be put together from files in a few places in the LightGBM source. We accomplished this with a script that just copies those files around.

```
.
├── CMakeLists.txt
├── LICENSE
├── R-package
│   ├── DESCRIPTION
│   ├── LICENSE
│   ├── NAMESPACE
│   ├── R
│   ├── README.md
│   ├── configure
│   ├── configure.ac
│   ├── configure.win
│   ├── data
│   ├── inst
│   ├── man
│   ├── pkgdown
│   ├── src
│   └── tests
├── README.md
├── compute
├── include
│   └── LightGBM
└── src
    ├── application
    ├── boosting
    ├── c_api.cpp
    ├── io
    ├── main.cpp
    ├── metric
    ├── network
    ├── objective
    └── treelearner
```





lightgbm_2.3.2.tar.gz

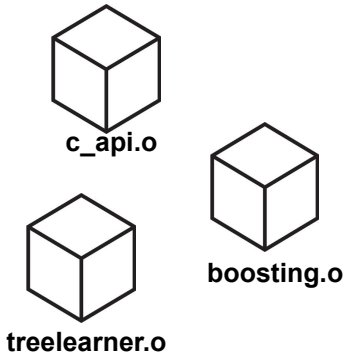
[properly-documented] {lightgbm}'s documentation had a number of issues like undocumented parameters, duplicated names, and typos.

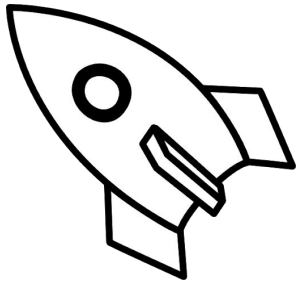
```
* checking Rd \usage sections ... WARNING
Undocumented arguments in documentation object 'lgb.cv'
  ‘save_name’
Duplicated \argument entries in documentation object 'lgb.cv':
  ‘params’ ‘data’ ‘nrounds’ ‘obj’ ‘boosting’ ‘num_leaves’ ‘max_depth’
  ‘num_threads’ ‘eval’ ‘verbose’ ‘record’ ‘eval_freq’ ‘init_model’
  ‘colnames’ ‘categorical_feature’ ‘early_stopping_rounds’ ‘callbacks’
  ‘...’
Documented arguments not in \usage in documentation object 'lgb.cv':
  ‘boosting’ ‘num_leaves’ ‘max_depth’ ‘num_threads’

Functions with \usage entries need to have the appropriate \alias
entries, and all their arguments documented.
The \usage entries must correspond to syntactically valid R code.
```

[self-contained] and [properly-documented] are the easy ones :)

[portable] Requires a quick introduction to building a C++ library

*.cpp, *.h





lib_lightgbm.so

1. You write code

2. A **compiler** is used to turn the code into objects. It does different stuff based on **compiler flags**.

3. A **linker** is used to connect all the objects, and symbols from libraries they use, into a new library. It does different stuff based on **linker flags**.

Figuring out which flags to use, and what order to build files, and a bunch of other nonsense, is handled by a **build tool**.

[portable, easy-to-install] A collection of those key things (build tools, compiler) is called a “toolchain”. {lightgbm} uses a different toolchain than CRAN today.

```
* installing *source* package 'lightgbm' ...
** using staged installation
** libs

*** arch - i386
installing to d:/RCompile/CRANguest/R-devel/lib/00LOCK-lightgbm/00new/lightgbm
[1] "Trying to build with MinGW"
Warning in system(paste0(cmake_cmd, " .. ")) : 'cmake' not found
Warning in system(paste0(cmake_cmd, " .. ")) : 'cmake' not found
Warning in system(build_cmd) : 'mingw32-make.exe' not found
Error in eval(ei, envir) : Cannot find lib_lightgbm.dll
* removing 'd:/RCompile/CRANguest/R-devel/lib/lightgbm'
```

{lightgbm} uses **CMake** to generate **GNU Make Makefiles**, which then compile the library with **MSVC**, **MinGW**, **gcc**, or **clang**. CMake is not supported on CRAN 😬😬😬😬

[portable, easy-to-install] To be sure {lightgbm} can be installed on many operating systems, we're using GNU Autoconf. Basically you write a weird shell script and then some magic turns it into a much bigger, more awesome shell script.

configure.ac

```
###·configure.ac·····-*-·Autoconf·-*-
#·Template·used·by·Autoconf·to·generate·'configure'·
#···*·https://unconj.ca/blog/an-autoconf-primer-for-
#···*·https://cran.r-project.org/doc/manuals/r-relea

AC_PREREQ(2.69)
AC_INIT([lightgbm],·[2.3.2],·[],·[lightgbm],·[])

############################
#·find·compiler·and·flags·#
############################

AC_MSG_CHECKING([location·of·R])
AC_MSG_RESULT([${R_HOME}])

#·set·up·CPP·flags
#·find·the·compiler·and·compiler·flags·used·by·R.
:·${R_HOME=`R·HOME`}
if·test·-z·"${R_HOME}";·then
····echo·"could·not·determine·R_HOME"
····exit·1
fi
CC=`"${R_HOME}/bin/R"·CMD·config·CC`
CXX=`"${R_HOME}/bin/R"·CMD·config·CXX11`
CFLAGS=`"${R_HOME}/bin/R"·CMD·config·CFLAGS`
CPPFLAGS=`"${R_HOME}/bin/R"·CMD·config·CPPFLAGS`
```

configure

```
#!·/bin/sh
#·Guess·values·for·system-dependent·variables·and·create·Makefiles.
#·Generated·by·GNU·Autoconf·2.69·for·lightgbm·2.3.2.
#
#
#·Copyright·(C)·1992-1996,·1998-2012·Free·Software·Foundation,·Inc.
#
#
#·This·configure·script·is·free·software;·the·Free·Software·Foundation
#·gives·unlimited·permission·to·copy,·distribute·and·modify·it.
##·--------------------·##
##·M4sh·Initialization.·##
##·--------------------·##

#·Be·more·Bourne·compatible
DUALCASE=1;·export·DUALCASE·#·for·MKS·sh
if·test·-n·"${ZSH_VERSION+set}"·&&·(emulate·sh)·>/dev/null·2>&1;·then·:
··emulate·sh
··NULLCMD=:
··#·Pre-4.2·versions·of·Zsh·do·word·splitting·on·${1+"$@"},·which
··#·is·contrary·to·our·usage.··Disable·this·feature.
··alias·-g·'${1+"$@"}'='"$@"'
··setopt·NO_GLOB_SUBST
else
··case·`(set·-o)·2>/dev/null`·in·#(
··*posix*)·:
····set·-o·posix·;;·#(
··*)·:
····;;
esac
fi
```

[portable, easy-to-install] That script created by autoconf runs whenever the package is installed, and changes a templated Makefile.

Makevars.in

```
CXX_STD = CXX11

PKGROOT=.

LGB_CPPFLAGS = \
    @LGB_CPPFLAGS@ \
    -DUSE_SOCKET \
    -DLGB_R_BUILD

CPICFLAGS = -fPIC

PKG_CPPFLAGS = \
    -I$(PKGROOT)/include \
    $(LGB_CPPFLAGS)
```

All the stuff in “@” signs gets replaced by the code in **configure**.

The stuff with “-D” means “define”, which can be used kind of like commenting/uncommenting the C++ source code.

```
static void Write(LogLevel level,
                  const char *level_str,
                  const char *format,
                  va_list val) {
  if (level <= GetLevel()) {  //omit the message with low level
//R code should write back to R's output stream,
//otherwise to stdout
#ifndef LGB_R_BUILD
    if (GetLogCallBack() == nullptr) {
      printf("[LightGBM] [%s] ", level_str);
      vprintf(format, val);
      printf("\n");
      fflush(stdout);
    } else {
      const size_t kBufSize = 512;
      char buf[kBufSize];
      snprintf(buf, kBufSize, "[LightGBM] [%s] ", level_str);
      GetLogCallBack()(buf);
      vsnprintf(buf, kBufSize, format, val);
      GetLogCallBack()(buf);
      GetLogCallBack()("\n");
    }
#else
    Rprintf("[LightGBM] [%s] ", level_str);
    Rvprintf(format, val);
    Rprintf("\n");
#endif
  }
```

Lesson: {lightgbm} drifted from CRAN practices because it is “an R wrapper for a project” not “an R package”. It’s a pattern you see in other similar projects

Even with support from RStudio, it took **{arrow}** a year to get to CRAN.

{xgboost} is on CRAN but only thanks to some heroic efforts by maintainers.

{RGF}, **{tensorflow}** got to CRAN by just using **{reticulate}** to call their Python packages that call C++ libraries.

{catboost} is still not on CRAN.

{xlearn} abandoned their R package 3 years ago.

Solution: Continuous Integration (CI) is a great communication tool to be sure the problems you fix stay fixed.

Like Big Bertha, a huge drill that builds a shell of concrete tiles behind itself as it digs.

Thank you to everyone on LightGBM for reviewing my pull requests and patiently teaching me new things

- @guolinke **Guolin Ke** (C++ code / R-package / Python-package)
- @chivee **Qiwei Ye** (C++ code / Python-package)
- @btrotta **Belinda Trotta** (C++ code)
- @Laurae2 **Damien Soukhavong** (R-package)
- @jameslamb **James Lamb** (R-package)
- @wxchan **Wenxuan Chen** (Python-package)
- @henry0312 **Tsukasa Omoto** (Python-package)
- @StrikerRUS **Nikita Titov** (Python-package)
- @huanzhang12 **Huan Zhang** (GPU support)

Come by https://github.com/microsoft/LightGBM , we're nice!

Thanks for your time!