# Machine Learning Mastery with R Mini-Course

## From Developer To R Machine Learning Practitioner in 14 Days

---

Jason Brownlee

MACHINE LEARNING MASTERY

Jason Brownlee

# Machine Learning Mastery with R Mini-Course

## From Developer To R Machine Learning Practitioner in 14 Days

**Machine Learning Mastery with R Mini-Course**



Edition: v1.0

Find the latest version of this guide online at: http://MachineLearningMastery.com

# Contents

# Before We Get Started...

In this mini-course you will discover how you can get started, build accurate models and confidently complete predictive modeling machine learning projects using R in 14 days.

**This is a long and useful guide. You might want to print it out.**

## Who Is This Mini-Course For?

Before we get started, let's make sure you are in the right place. The list below provides some general guidelines as to who this course was designed for. Don't panic if you don't match these points exactly, you might just need to brush up in one area or another to keep up.

- **Developers that know how to write a little code**. This means that it is not a big deal for you to pick up a new programming language like R once you know the basic syntax. It does not mean your a wizard coder, just that you can follow a basic C-like language with little effort.
- **Developers that know a little machine learning**. This means you know about the basics of machine learning like cross-validation, some algorithms and bias-variance trade-off. It does not mean that you are a machine learning PhD, just that you know the landmarks or know where to look them up.

This mini-course is neither a textbook on R or a textbook on machine learning. It will take you from a developer that knows a little machine learning to a developer who can get results using R, the most powerful and most popular platform for machine learning.

## Mini-Course Overview (what to expect)

This mini-course is broken down into 14 lessons that I call *days*. You could complete one lesson per day (recommended) or complete all of the lessons in one day (hard core!). It really depends on the time you have available and your level of enthusiasm. Below are 14 lessons that will get you started and productive with machine learning in R:

- **Day 1**: Download and Install R.
- **Day 2**: Get Around In R with Basic Syntax.
- **Day 3**: Load Data and Standard Machine Learning Datasets.

- **Day 4**: Understand Data with Descriptive Statistics.
- **Day 5**: Understand Data with Visualization.
- **Day 6**: Prepare For Modeling by Pre-Processing Data.
- **Day 7**: Algorithm Evaluation With Resampling Methods.
- **Day 8**: Algorithm Evaluation Metrics.
- **Day 9**: Spot-Check Algorithms.
- **Day 10**: Model Comparison and Selection.
- **Day 11**: Improve Accuracy with Algorithm Tuning.
- **Day 12**: Improve Accuracy with Ensemble Predictions.
- **Day 13**: Finalize And Save Your Model.
- **Day 14**: Hello World End-to-End Project.

Each lesson could take you 60 seconds or up to 30 minutes. Take your time and complete the lessons at your own pace. The lessons expect you to go off and find out how to do things. I will give you hints, but part of the point of each lesson is to force you to learn where to go to look for help on and about the R platform (hint, I have all of the answers directly on my blog[1], use the search). I do provide more help in the early lessons because I want you to build up some confidence and inertia.

**Hang in there, don't give up!**

If you would like me to step you through each lesson in great detail, take a look at my book:
**Machine Learning Mastery with R:**
https://machinelearningmastery.com/machine-learning-with-r/

[1] http://MachineLearningMastery.com

# Day 1: Download and Install R

You cannot get started with machine learning in R until you have access to the platform. Today's lesson is easy, you must download and install the R platform on your computer.

1. Visit the R homepage and download R for your operating system: https://www.r-project.org.
2. Install R on your computer.
3. Start R for the first time from command line by typing `R`.

If you need help installing R, checkout the post: *Use R For Machine Learning*[2].

[2] http://machinelearningmastery.com/use-r-for-machine-learning

# Day 2: Try Some Basic R Syntax

You need to be able to read and write basic R scripts. As a developer you can pick-up new programming languages pretty quickly. R is case sensitive, uses hash (#) for comments and uses the arrow operator (<-) for assignments instead of the single equals (=). Today's task is to practice the basic syntax of the R programming language in the R interactive environment.

1. Practice assignment in the language using the arrow operator (<-).
2. Practice using basic data structures like vectors, lists and data frames.
3. Practice using flow control structures like If-Then-Else and loops.
4. Practice calling functions, installing and loading packages.

For example, below is an example of creating a list of numbers and calculating the mean.

```
numbers <- c(1, 2, 3, 4, 5, 6)
mean(numbers)
```

Listing 1: Calculate the mean of a vector of numbers.

If you need help with basic R syntax, see the post: *Super Fast Crash Course in R*[3].

[3] http://machinelearningmastery.com/r-crash-course-for-developers

# Day 3: Load Standard Datasets

Machine learning algorithms need data. You can load your own data from CSV files but when you are getting started with machine learning in R you should practice on standard machine learning datasets. Your task for todays lesson are to get comfortable loading data into R and to find and load standard machine learning datasets.

The datasets package that comes with R has many standard datasets including the famous iris flowers dataset. The `mlbench` package also contains man standard machine learning datasets.

1. Practice loading CSV files into R using the `read.csv()` function.
2. Practice loading standard machine learning datasets from the `datasets` and `mlbench` packages.

You can get help about a function by typing `?FunctionName` or by calling the `help()` function and passing the function name that you need help with as an argument. To get you started, the below snippet will install and load the `mlbench` package, list all of the datasets it offers and attach the `PimaIndiansDiabetes` dataset to your environment for you to play with.

```
install.packages("mlbench")
library(mlbench)
data(package="mlbench")
data(PimaIndiansDiabetes)
head(PimaIndiansDiabetes)
```

Listing 2: Install and load the mlbench package and attach a dataset.

Well done for making it this far! Hang in there.

# Day 4: Understand Data with Descriptive Statistics

Once you have loaded your data into R you need to be able to understand it. The better you can understand your data, the better and more accurate the models that you can build. The first step to understanding your data is to use descriptive statistics. Today your lesson is to learn how to use descriptive statistics to understand your data.

1. Understand your data using the `head()` function to look at the first few rows.
2. Review the dimensions of your data with the `dim()` function.
3. Review the distribution of your data with the `summary()` function.
4. Calculate pairwise correlation between your variables using the `cor()` function.

The below example loads the iris dataset and summarizes the distribution of each attribute.

```
1 data(iris)
2 summary(iris)
```

Listing 3: Attach and summarize the iris flowers dataset.

Try it out!

# Day 5: Understand Data with Visualization

Continuing on from yesterdays lesson, you must spend time to better understand your data. A second way to improve your understanding of your data is by using data visualization techniques (e.g. plotting). Today, your lesson is to learn how to use plotting in R to understand attributes alone and their interactions.

1. Use the `hist()` function to create a histogram of each attribute.
2. Use the `boxplot()` function to create box and whisker plots of each attribute.
3. Use the `pairs()` function to create pairwise scatter plots of all attributes.

For example the snippet below will load the iris dataset and create a scatter plot matrix of the dataset.

```
1 data(iris)
2 pairs(iris)
```

Listing 4: Create a scatter plot matrix of the iris dataset attributes.

# Day 6: Prepare For Modeling by Pre-Processing Data

Your raw data may not be setup to be in the best shape for modeling. Sometimes you need to pre-process your data in order to best present the inherent structure of the problem in your data to the modeling algorithms. In today's lesson, you will use the pre-processing capabilities provided by the caret package.

The `caret` package provides the `preprocess()` function that takes a `method` argument to indicate the type of pre-processing to perform. Once the pre-processing parameters have been prepared from a dataset, the same pre-processing step can be applied to each dataset that you may have. Remember, you can install and load the `caret` package as follows:

```
install.packages("caret")
library(caret)
```

Listing 5: Install and load the caret package.

1. Standardize numerical data (e.g. mean of 0 and standard deviation of 1) using the `scale` and `center` options.
2. Normalize numerical data (e.g. to a range of 0-1) using the `range` option.
3. Explore more advanced power transforms like the Box-Cox power transform with the `BoxCox` option.

For example, the snippet below loads the iris dataset, calculates the parameters needed to normalize the data, then creates a normalized copy of the data.

```
# load caret package
library(caret)
# load the dataset
data(iris)
# calculate the pre-process parameters from the dataset
preprocessParams <- preProcess(iris[,1:4], method=c("range"))
# transform the dataset using the pre-processing parameters
transformed <- predict(preprocessParams, iris[,1:4])
# summarize the transformed dataset
summary(transformed)
```

Listing 6: Normalize the numeric attributes in the iris dataset.

# Day 7: Algorithm Evaluation With Resampling Methods

The dataset used to train a machine learning algorithm is called a training dataset. The dataset used to train an algorithm cannot be used to give you reliable estimates of the accuracy of the model on new data. This is a big problem because the whole idea of creating the model is to make predictions on new data.

You can use statistical methods called resampling methods to split your training dataset up into subsets, some are used to train the model and others are held back and used to estimate the accuracy of the model on unseen data. Your goal with todays lesson is to practice using the different resampling methods available in the caret package. Look up the help on the `createDataPartition()`, `trainControl()` and `train()` functions in R.

1. Split a dataset into training and test sets.
2. Estimate the accuracy of an algorithm using $k$-fold cross-validation.
3. Estimate the accuracy of an algorithm using repeated $k$-fold cross-validation.

The snippet below uses the caret package to estimate the accuracy of the Naive Bayes algorithm on the iris dataset using 10-fold cross-validation.

```
# load the library
library(caret)
# load the iris dataset
data(iris)
# define training control
trainControl <- trainControl(method="cv", number=10)
# estimate the accuracy of Naive Bayes on the dataset
fit <- train(Species~., data=iris, trControl=trainControl, method="nb")
# summarize the estimated accuracy
print(fit)
```

Listing 7: Estimate the accuracy of Naive Bayes using 10-fold cross-validation.

Did you realize that this is the half-way point? Well done!

# Day 8: Algorithm Evaluation Metrics

There are many different metrics that you can use to evaluate the skill of a machine learning algorithm on a dataset. You can specify the metric used for your test harness with `caret` in the `train()` function and defaults can be used for regression and classification problems. Your goal with today's lesson is to practice using the different algorithm performance metrics available in the `caret` package.

1. Practice using the Accuracy and Kappa metrics on a classification problem (e.g. `iris` dataset).
2. Practice using RMSE and RSquared metrics on a regression problem (e.g. `longley` dataset).
3. Practice using the ROC metrics on a binary classification problem (e.g. `PimaIndiansDiabetes` dataset from the `mlbench` package).

The snippet below demonstrates calculating the `logLoss` metric on the `iris` dataset.

```
# load caret library
library(caret)
# load the iris dataset
data(iris)
# prepare 5-fold cross-validation and keep the class probabilities
control <- trainControl(method="cv", number=5, classProbs=TRUE, summaryFunction=mnLogLoss)
# estimate accuracy using LogLoss of the CART algorithm
fit <- train(Species~., data=iris, method="rpart", metric="logLoss", trControl=control)
# display results
print(fit)
```

Listing 8: Evaluate CART using LogLoss metric on the iris dataset.

# Day 9: Spot-Check Machine Learning Algorithms

You cannot possibly know which algorithm will perform best on your data beforehand. You have to discover it using a process of trial and error. I call this spot-checking algorithms. The caret package provides an interface to many machine learning algorithms and tools to compare the estimated accuracy of those algorithms. In this lesson you must practice spot-checking different machine learning algorithms.

1. Spot-check linear algorithms on a dataset (e.g. linear regression, logistic regression and linear discriminate analysis).
2. Spot-check some nonlinear algorithms on a dataset (e.g. KNN, SVM and CART).
3. Spot-check some sophisticated ensemble algorithms on a dataset (e.g. random forest and stochastic gradient boosting).

You can get a list of models that you can use in caret by typing: `names(getModelInfo())`. For example, the snippet below spot-checks two linear algorithms on the Pima Indians Diabetes dataset from the `mlbench` package.

```
# load libraries
library(caret)
library(mlbench)
# load the Pima Indians Diabetes dataset
data(PimaIndiansDiabetes)
# prepare 10-fold cross-validation
trainControl <- trainControl(method="cv", number=10)
# estimate accuracy of logistic regression
set.seed(7)
fit.lr <- train(diabetes~., data=PimaIndiansDiabetes, method="glm", trControl=trainControl)
# estimate accuracy of linear discriminate analysis
set.seed(7)
fit.lda <- train(diabetes~., data=PimaIndiansDiabetes, method="lda", trControl=trainControl)
# collect resampling statistics
results <- resamples(list(LR=fit.lr, LDA=fit.lda))
# summarize results
summary(results)
```

Listing 9: Spot-check two algorithms on the Pima Indians Diabetes dataset.

# Day 10: Model Comparison and Selection

Now that you know how to spot-check machine learning algorithms on your dataset, you need to know how to compare the estimated performance of different algorithms and select the best model. Thankfully the caret package provides a suite of tools to plot and summarize the differences in performance between models. In today's lesson you will practice comparing the accuracy of machine learning algorithms in R.

- Use the `summary()` caret function to create a table of results (hint, there is an example in the previous lesson)
- Use the `dotplot()` caret function to compare results.
- Use the `bwplot()` caret function to compare results.
- Use the `diff()` caret function to calculate the statistical significance between results.

The snippet below extends yesterdays example and creates a plot of the spot-check results.

```
# load libraries
library(caret)
library(mlbench)
# load the Pima Indians Diabetes dataset
data(PimaIndiansDiabetes)
# prepare 10-fold cross-validation
trainControl <- trainControl(method="cv", number=10)
# estimate accuracy of logistic regression
set.seed(7)
fit.lr <- train(diabetes~., data=PimaIndiansDiabetes, method="glm", trControl=trainControl)
# estimate accuracy of linear discriminate analysis
set.seed(7)
fit.lda <- train(diabetes~., data=PimaIndiansDiabetes, method="lda", trControl=trainControl)
# collect resampling statistics
results <- resamples(list(LR=fit.lr, LDA=fit.lda))
# plot the results
dotplot(results)
```

Listing 10: Evaluate two algorithms on the Pima Indians Diabetes dataset.

# Day 11: Improve Accuracy with Algorithm Tuning

Once you have found one or two algorithms that perform well on your dataset, you may want to improve the performance of those models. One way to increase the performance of an algorithm is to tune it's parameters to your specific dataset. The `caret` package provides three ways to search for combinations of parameters for a machine learning algorithm. Your goal in today's lesson is to practice each.

1. Tune the parameters of an algorithm automatically (e.g. see the `tuneLength` argument to `train()`).
2. Tune the parameters of an algorithm using a grid search that you specify.
3. Tune the parameters of an algorithm using a random search.

Take a look at the help for the `trainControl()` and `train()` functions and take note of the `method` and the `tuneGrid` arguments. The snippet below uses is an example of using a grid search for the random forest algorithm (`rf`) on the iris dataset.

```
# load the library
library(caret)
# load the iris dataset
data(iris)
# define training control
trainControl <- trainControl(method="cv", number=10)
# define a grid of parameters to search for random forest
grid <- expand.grid(.mtry=c(1,2,3,4,5,6,7,8,10))
# estimate the accuracy of Random Forest on the dataset
fit <- train(Species~., data=iris, trControl=trainControl, tuneGrid=grid, method="rf")
# summarize the estimated accuracy
print(fit)
```

Listing 11: Example tuning the Random Forest algorithm with a grid search.

You're nearly at the end! Just a few more lessons to go!

# Day 12: Improve Accuracy with Ensemble Predictions

Another way that you can improve the performance of your models is to combine the predictions from multiple models. Some models provide this capability built-in such as random forest for bagging and stochastic gradient boosting for boosting. Another type of ensembling called stacking (or blending) can learn how to best combine the predictions from multiple models and is provided in the package `caretEnsemble`. In today's lesson you will practice using ensemble methods.

1. Practice bagging ensembles with the random forest and bagged CART algorithms in caret.
2. Practice boosting ensembles with the gradient boosting machine and C5.0 algorithms in caret.
3. Practice stacking ensembles using the `caretEnsemble` package and the `caretStack()` function.

The snippet below demonstrates how you can combine the predictions from multiple models using stacking.

```
# Load packages
library(mlbench)
library(caret)
library(caretEnsemble)
# load the Pima Indians Diabetes dataset
data(PimaIndiansDiabetes)
# create sub-models
trainControl <- trainControl(method="cv", number=5, savePredictions=TRUE, classProbs=TRUE)
algorithmList <- c('knn', 'glm')
set.seed(7)
models <- caretList(diabetes~., data=PimaIndiansDiabetes, trControl=trainControl,
    methodList=algorithmList)
print(models)
# learn how to best combine the predictions
stackControl <- trainControl(method="cv", number=5, savePredictions=TRUE, classProbs=TRUE)
set.seed(7)
stack.glm <- caretStack(models, method="glm", trControl=stackControl)
print(stack.glm)
```

Listing 12: Example of combining predictions using stacking ensemble.

# Day 13: Finalize And Save Your Model

Once you have found a well performing model on your machine learning problem, you need to finalize it. In today's lesson you will practice the tasks related to finalizing your model.

1. Practice using the `predict()` function to make predictions with a model trained using `caret`.
2. Practice training standalone versions of well performing models.
3. Practice saving trained models to file and loading them up again using the `saveRDS()` and `readRDS()` functions.

For example, the snippet below shows how you can create a random forest algorithm trained on your entire dataset ready for general use.

```
# load package
library(randomForest)
# load iris data
data(iris)
# train random forest model
finalModel <- randomForest(Species~., iris, mtry=2, ntree=2000)
# display the details of the final model
print(finalModel)
```

Listing 13: Example of creating a standalone random forest algorithm.

# Day 14: Hello World End-to-End Project

You now know how to complete each task of a predictive modeling machine learning problem. In todays lesson you need to practice putting the pieces together and working through a standard machine learning dataset end-to-end.

1. Work through the iris dataset end-to-end (the *hello world* of machine learning)

This includes the steps:

1. Understanding your data using descriptive statistics and visualization.
2. Pre-Processing the data to best expose the structure of the problem.
3. Spot-checking a number of algorithms using your own test harness.
4. Improving results using algorithm parameter tuning.
5. Improving results using ensemble methods.
6. Finalize the model ready for future use.

# Final Word Before You Go...

*You made it. Well done!* Take a moment and look back at how far you have come:

- You started off with a strong desire to be able to practice machine learning using R.
- You downloaded, installed and started R, perhaps for the first time.
- Slowly and steadily you learned how the standard tasks of a predictive modeling machine learning project map onto the R platform.
- Building upon the recipes for common machine learning tasks you worked through your first machine learning problems end-to-end using R.
- Using the skills you have developed you are now capable of working through new and different predictive modeling machine learning problems on your own.

Don't make light of this, you have come a long way in a short amount of time. This is just the beginning of your machine learning journey with R. Keep practicing and developing your skills.

## How Did You Go With The Mini-Course?

Did you enjoy this mini-course?
Do you have any questions or sticking points?

Let me know, send me an email at: **jason@MachineLearningMastery.com**

P.S. If you are looking to take the next step, take a look at my book:
**Machine Learning Mastery with R**
https://machinelearningmastery.com/machine-learning-with-r/