This is code that will encompany an article that will appear in a special edition of a German IT magazine. The article is about explaining black-box machine learning models. In that article I’m showcasing three practical examples:

1. Explaining supervised classification models built on tabular data using caret and the iml package
2. Explaining image classification models with keras and lime
3. Explaining text classification models with xgboost and lime

Below, you will find the code:

# data wrangling

library(tidyverse)

library(readr)

# plotting

library(ggthemes)

theme\_set(theme\_minimal())

# text prep

library(text2vec)

# ml

library(caret)

library(xgboost)

# explanation

library(lime)

**Text classification models**

Here I am using another Kaggle dataset: [Women’s e-commerce cloting reviews](https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews). The data contains a text review of different items of clothing, as well as some additional information, like rating, division, etc.

In this example, I will use the review title and text in order to classify whether or not the item was liked. I am creating the response variable from the rating: every item rates with 5 stars is considered “liked” (1), the rest as “not liked” (0). I am also combining review title and text.

clothing\_reviews <- read\_csv("/Users/shiringlander/Documents/Github/ix\_lime\_etc/Womens Clothing E-Commerce Reviews.csv") %>%

mutate(Liked = as.factor(ifelse(Rating == 5, 1, 0)),

text = paste(Title, `Review Text`),

text = gsub("NA", "", text))

## Parsed with column specification:

## cols(

## X1 = col\_integer(),

## `Clothing ID` = col\_integer(),

## Age = col\_integer(),

## Title = col\_character(),

## `Review Text` = col\_character(),

## Rating = col\_integer(),

## `Recommended IND` = col\_integer(),

## `Positive Feedback Count` = col\_integer(),

## `Division Name` = col\_character(),

## `Department Name` = col\_character(),

## `Class Name` = col\_character()

## )

glimpse(clothing\_reviews)

## Observations: 23,486

## Variables: 13

## $ X1 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...

## $ `Clothing ID` 767, 1080, 1077, 1049, 847, 1080, 85...

## $ Age 33, 34, 60, 50, 47, 49, 39, 39, 24, ...

## $ Title NA, NA, "Some major design flaws", "...

## $ `Review Text` "Absolutely wonderful - silky and se...

## $ Rating 4, 5, 3, 5, 5, 2, 5, 4, 5, 5, 3, 5, ...

## $ `Recommended IND` 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, ...

## $ `Positive Feedback Count` 0, 4, 0, 0, 6, 4, 1, 4, 0, 0, 14, 2,...

## $ `Division Name` "Initmates", "General", "General", "...

## $ `Department Name` "Intimate", "Dresses", "Dresses", "B...

## $ `Class Name` "Intimates", "Dresses", "Dresses", "...

## $ Liked 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, ...

## $ text " Absolutely wonderful - silky and s...

Whether an item was liked or not will thus be my response variable or label for classification.

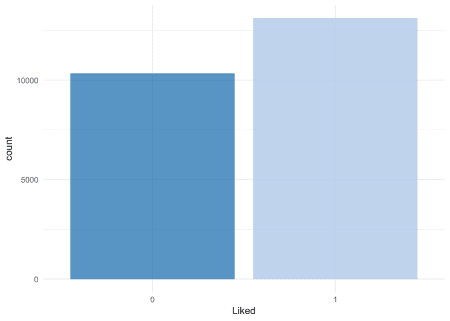
clothing\_reviews %>%

ggplot(aes(x = Liked, fill = Liked)) +

geom\_bar(alpha = 0.8) +

scale\_fill\_tableau(palette = "tableau20") +

guides(fill = FALSE)



Let’s split the data into train and test sets:

set.seed(42)

idx <- createDataPartition(clothing\_reviews$Liked,

p = 0.8,

list = FALSE,

times = 1)

clothing\_reviews\_train <- clothing\_reviews[ idx,]

clothing\_reviews\_test <- clothing\_reviews[-idx,]

**Let’s start simple**

The first text model I’m looking at has been built similarly to the example model in the help for lime::interactive\_text\_explanations().

First, we need to prepare the data for modeling: we will need to convert the text to a document term matrix (dtm). There are different ways to do this. One is be with the text2vec package.

“Because of R’s copy-on-modify semantics, it is not easy to iteratively grow a DTM. Thus constructing a DTM, even for a small collections of documents, can be a serious bottleneck for analysts and researchers. It involves reading the whole collection of text documents into RAM and processing it as single vector, which can easily increase memory use by a factor of 2 to 4. The text2vec package solves this problem by providing a better way of constructing a document-term matrix.”

Alternatives to text2vec would be tm + SnowballC or you could work with the tidytext package.

The itoken() function creates vocabularies (here stemmed words), from which we can create the dtm with the create\_dtm() function.

All preprocessing steps, starting from the raw text, need to be wrapped in a function that can then be pasted into the lime::lime() function; this is only necessary if you want to use your model with lime.

get\_matrix <- function(text) {

it <- itoken(text, progressbar = FALSE)

create\_dtm(it, vectorizer = hash\_vectorizer())

}

Now, this preprocessing function can be applied to both training and test data.

dtm\_train <- get\_matrix(clothing\_reviews\_train$text)

str(dtm\_train)

## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots

## ..@ i : int [1:889012] 304 764 786 788 793 794 1228 2799 2819 3041 ...

## ..@ p : int [1:262145] 0 0 0 0 0 0 0 0 0 0 ...

## ..@ Dim : int [1:2] 18789 262144

## ..@ Dimnames:List of 2

## .. ..$ : chr [1:18789] "1" "2" "3" "4" ...

## .. ..$ : NULL

## ..@ x : num [1:889012] 1 1 2 1 2 1 1 1 1 1 ...

## ..@ factors : list()

dtm\_test <- get\_matrix(clothing\_reviews\_test$text)

str(dtm\_test)

## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots

## ..@ i : int [1:222314] 2793 400 477 622 2818 2997 3000 4500 3524 2496 ...

## ..@ p : int [1:262145] 0 0 0 0 0 0 0 0 0 0 ...

## ..@ Dim : int [1:2] 4697 262144

## ..@ Dimnames:List of 2

## .. ..$ : chr [1:4697] "1" "2" "3" "4" ...

## .. ..$ : NULL

## ..@ x : num [1:222314] 1 1 1 1 1 1 1 1 1 1 ...

## ..@ factors : list()

And we use it to train a model with the xgboost package (just as in the example of the lime package).

xgb\_model <- xgb.train(list(max\_depth = 7,

eta = 0.1,

objective = "binary:logistic",

eval\_metric = "error", nthread = 1),

xgb.DMatrix(dtm\_train,

label = clothing\_reviews\_train$Liked == "1"),

nrounds = 50)

Let’s try it on the test data and see how it performs:

pred <- predict(xgb\_model, dtm\_test)

confusionMatrix(clothing\_reviews\_test$Liked,

as.factor(round(pred, digits = 0)))

## Confusion Matrix and Statistics

##

## Reference

## Prediction 0 1

## 0 1370 701

## 1 421 2205

##

## Accuracy : 0.7611

## 95% CI : (0.7487, 0.7733)

## No Information Rate : 0.6187

## P-Value [Acc > NIR] : < 2.2e-16

##

## Kappa : 0.5085

## Mcnemar's Test P-Value : < 2.2e-16

##

## Sensitivity : 0.7649

## Specificity : 0.7588

## Pos Pred Value : 0.6615

## Neg Pred Value : 0.8397

## Prevalence : 0.3813

## Detection Rate : 0.2917

## Detection Prevalence : 0.4409

## Balanced Accuracy : 0.7619

##

## 'Positive' Class : 0

##

Okay, not a perfect score but good enough for me – right now, I’m more interested in the explanations of the model’s predictions. For this, we need to run the lime() function and give it

* the text input that was used to construct the model
* the trained model
* the preprocessing function

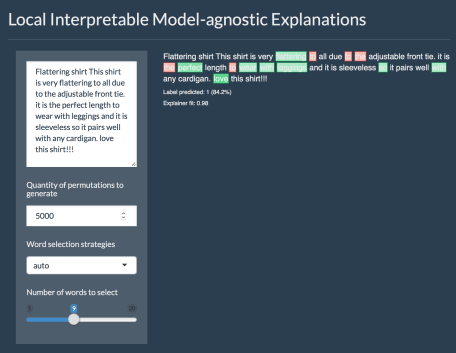
explainer <- lime(clothing\_reviews\_train$text,

xgb\_model,

preprocess = get\_matrix)

With this, we could right away call the interactive explainer Shiny app, where we can type any text we want into the field on the left and see the explanation on the right: words that are underlined green support the classification, red words contradict them.

interactive\_text\_explanations(explainer)



What happens in the background in the app, we can do explicitly by calling the explain() function and give it

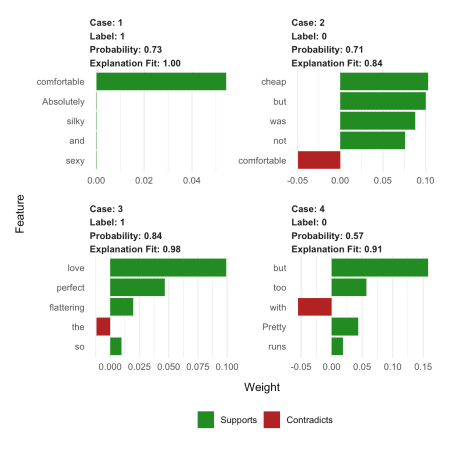
* the test data (here the first four reviews of the test set)
* the explainer defined with the lime() function
* the number of labels we want to have explanations for (alternatively, you set the label by name)
* and the number of features (in this case words) that should be included in the explanations

We can plot them either with the plot\_text\_explanations() function, which gives an output like in the Shiny app or we use the regular plot\_features() function.

explanations <- lime::explain(clothing\_reviews\_test$text[1:4], explainer, n\_labels = 1, n\_features = 5)

plot\_text\_explanations(explanations)

plot\_features(explanations)



As we can see, our explanations contain a lot of stop-words that don’t really make much sense as features in our model. So…

**… let’s try a more complex example**

Okay, our model above works but there are still common words and stop words in our model that LIME picks up on. Ideally, we would want to remove them before modeling and keep only relevant words. This we can accomplish by using additional steps and options in our preprocessing function.

Important to know is that whatever preprocessing we do with our text corpus, train and test data has to have the same features (i.e. words)! If we were to incorporate all the steps shown below into one function and call it separately on train and test data, we would end up with different words in our dtm and the predict() function won’t work any more. In the simple example above:

Nevertheless, the lime::explain() function expects a preprocessing function that takes a character vector as input.

How do we go about this? First, we will need to create the vocabulary just from the training data. To reduce the number of words to only the most relevant I am performing the following steps:

* stem all words
* remove step-words
* prune vocabulary
* transform into vector space

stem\_tokenizer <- function(x) {

lapply(word\_tokenizer(x),

SnowballC::wordStem,

language = "en")

}

stop\_words = tm::stopwords(kind = "en")

# create prunded vocabulary

vocab\_train <- itoken(clothing\_reviews\_train$text,

preprocess\_function = tolower,

tokenizer = stem\_tokenizer,

progressbar = FALSE)

v <- create\_vocabulary(vocab\_train,

stopwords = stop\_words)

pruned\_vocab <- prune\_vocabulary(v,

doc\_proportion\_max = 0.99,

doc\_proportion\_min = 0.01)

vectorizer\_train <- vocab\_vectorizer(pruned\_vocab)

This vector space can now be added to the preprocessing function, which we can then apply to both train and test data. Here, I am also transforming the word counts to [tfidf](https://en.wikipedia.org/wiki/Tf%E2%80%93idf) values.

# preprocessing function

create\_dtm\_mat <- function(text, vectorizer = vectorizer\_train) {

vocab <- itoken(text,

preprocess\_function = tolower,

tokenizer = stem\_tokenizer,

progressbar = FALSE)

dtm <- create\_dtm(vocab,

vectorizer = vectorizer)

tfidf = TfIdf$new()

fit\_transform(dtm, tfidf)

}

dtm\_train2 <- create\_dtm\_mat(clothing\_reviews\_train$text)

str(dtm\_train2)

## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots

## ..@ i : int [1:415770] 26 74 169 294 588 693 703 708 727 759 ...

## ..@ p : int [1:506] 0 189 380 574 765 955 1151 1348 1547 1740 ...

## ..@ Dim : int [1:2] 18789 505

## ..@ Dimnames:List of 2

## .. ..$ : chr [1:18789] "1" "2" "3" "4" ...

## .. ..$ : chr [1:505] "ad" "sandal" "depend" "often" ...

## ..@ x : num [1:415770] 0.177 0.135 0.121 0.17 0.131 ...

## ..@ factors : list()

dtm\_test2 <- create\_dtm\_mat(clothing\_reviews\_test$text)

str(dtm\_test2)

## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots

## ..@ i : int [1:103487] 228 304 360 406 472 518 522 624 732 784 ...

## ..@ p : int [1:506] 0 53 113 151 186 216 252 290 323 360 ...

## ..@ Dim : int [1:2] 4697 505

## ..@ Dimnames:List of 2

## .. ..$ : chr [1:4697] "1" "2" "3" "4" ...

## .. ..$ : chr [1:505] "ad" "sandal" "depend" "often" ...

## ..@ x : num [1:103487] 0.263 0.131 0.135 0.109 0.179 ...

## ..@ factors : list()

And we will train another gradient boosting model:

xgb\_model2 <- xgb.train(params = list(max\_depth = 10,

eta = 0.2,

objective = "binary:logistic",

eval\_metric = "error", nthread = 1),

data = xgb.DMatrix(dtm\_train2,

label = clothing\_reviews\_train$Liked == "1"),

nrounds = 500)

pred2 <- predict(xgb\_model2, dtm\_test2)

confusionMatrix(clothing\_reviews\_test$Liked,

as.factor(round(pred2, digits = 0)))

## Confusion Matrix and Statistics

##

## Reference

## Prediction 0 1

## 0 1441 630

## 1 426 2200

##

## Accuracy : 0.7752

## 95% CI : (0.763, 0.787)

## No Information Rate : 0.6025

## P-Value [Acc > NIR] : < 2.2e-16

##

## Kappa : 0.5392

## Mcnemar's Test P-Value : 4.187e-10

##

## Sensitivity : 0.7718

## Specificity : 0.7774

## Pos Pred Value : 0.6958

## Neg Pred Value : 0.8378

## Prevalence : 0.3975

## Detection Rate : 0.3068

## Detection Prevalence : 0.4409

## Balanced Accuracy : 0.7746

##

## 'Positive' Class : 0

##

Unfortunately, this didn’t really improve the classification accuracy but let’s look at the explanations again:

explainer2 <- lime(clothing\_reviews\_train$text,

xgb\_model2,

preprocess = create\_dtm\_mat)

explanations2 <- lime::explain(clothing\_reviews\_test$text[1:4], explainer2, n\_labels = 1, n\_features = 4)

plot\_text\_explanations(explanations2)

The words that get picked up now make much more sense! So, even though making my model more complex didn’t improve “the numbers”, this second model is likely to be much better able to generalize to new reviews because it seems to pick up on words that make intuitive sense.

That’s why I’m sold on the benefits of adding explainer functions to most machine learning workflows – and why I love the lime package in R!

sessionInfo()

## R version 3.5.1 (2018-07-02)

## Platform: x86\_64-apple-darwin15.6.0 (64-bit)

## Running under: macOS High Sierra 10.13.6

##

## Matrix products: default

## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib

## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib

##

## locale:

## [1] de\_DE.UTF-8/de\_DE.UTF-8/de\_DE.UTF-8/C/de\_DE.UTF-8/de\_DE.UTF-8

##

## attached base packages:

## [1] stats graphics grDevices utils datasets methods base

##

## other attached packages:

## [1] bindrcpp\_0.2.2 lime\_0.4.0 xgboost\_0.71.2 caret\_6.0-80

## [5] lattice\_0.20-35 text2vec\_0.5.1 ggthemes\_3.5.0 forcats\_0.3.0

## [9] stringr\_1.3.1 dplyr\_0.7.6 purrr\_0.2.5 readr\_1.1.1

## [13] tidyr\_0.8.1 tibble\_1.4.2 ggplot2\_3.0.0 tidyverse\_1.2.1

##

## loaded via a namespace (and not attached):

## [1] colorspace\_1.3-2 class\_7.3-14 rprojroot\_1.3-2

## [4] futile.logger\_1.4.3 pls\_2.6-0 rstudioapi\_0.7

## [7] DRR\_0.0.3 SnowballC\_0.5.1 prodlim\_2018.04.18

## [10] lubridate\_1.7.4 xml2\_1.2.0 codetools\_0.2-15

## [13] splines\_3.5.1 mnormt\_1.5-5 robustbase\_0.93-1

## [16] knitr\_1.20 shinythemes\_1.1.1 RcppRoll\_0.3.0

## [19] mlapi\_0.1.0 jsonlite\_1.5 broom\_0.4.5

## [22] ddalpha\_1.3.4 kernlab\_0.9-26 sfsmisc\_1.1-2

## [25] shiny\_1.1.0 compiler\_3.5.1 httr\_1.3.1

## [28] backports\_1.1.2 assertthat\_0.2.0 Matrix\_1.2-14

## [31] lazyeval\_0.2.1 cli\_1.0.0 later\_0.7.3

## [34] formatR\_1.5 htmltools\_0.3.6 tools\_3.5.1

## [37] NLP\_0.1-11 gtable\_0.2.0 glue\_1.2.0

## [40] reshape2\_1.4.3 Rcpp\_0.12.17 slam\_0.1-43

## [43] cellranger\_1.1.0 nlme\_3.1-137 blogdown\_0.6

## [46] iterators\_1.0.9 psych\_1.8.4 timeDate\_3043.102

## [49] gower\_0.1.2 xfun\_0.3 rvest\_0.3.2

## [52] mime\_0.5 stringdist\_0.9.5.1 DEoptimR\_1.0-8

## [55] MASS\_7.3-50 scales\_0.5.0 ipred\_0.9-6

## [58] hms\_0.4.2 promises\_1.0.1 parallel\_3.5.1

## [61] lambda.r\_1.2.3 yaml\_2.1.19 rpart\_4.1-13

## [64] stringi\_1.2.3 foreach\_1.4.4 e1071\_1.6-8

## [67] lava\_1.6.2 geometry\_0.3-6 rlang\_0.2.1

## [70] pkgconfig\_2.0.1 evaluate\_0.10.1 bindr\_0.1.1

## [73] labeling\_0.3 recipes\_0.1.3 htmlwidgets\_1.2

## [76] CVST\_0.2-2 tidyselect\_0.2.4 plyr\_1.8.4

## [79] magrittr\_1.5 bookdown\_0.7 R6\_2.2.2

## [82] magick\_1.9 dimRed\_0.1.0 pillar\_1.2.3

## [85] haven\_1.1.2 foreign\_0.8-70 withr\_2.1.2

## [88] survival\_2.42-3 abind\_1.4-5 nnet\_7.3-12

## [91] modelr\_0.1.2 crayon\_1.3.4 futile.options\_1.0.1

## [94] rmarkdown\_1.10 grid\_3.5.1 readxl\_1.1.0

## [97] data.table\_1.11.4 ModelMetrics\_1.1.0 digest\_0.6.15

## [100] tm\_0.7-4 xtable\_1.8-2 httpuv\_1.4.4.2

## [103] RcppParallel\_4.4.0 stats4\_3.5.1 munsell\_0.5.0

## [106] glmnet\_2.0-16 magic\_1.5-8