CREDIT SCORES

Reuben

2024-11-26

## Analyzing Credit Scores with tidymodels in R

Welcome to Analyzing Credit Scores with tidymodels in R!

In this live training, we’ll explore what differentiates consumer credit score levels and demonstrate how dimensionality reduction can retain much of the information in a dataset while reducing its size. We’ll use the embed and tidymodels to build UMAP and decision tree models. We will to demonstrate the concept of information by comparing the performance of decision tree models before and after applying UMAP dimensionality reduction.

### Setup Environment

First, we’ll load the necessary packages – tidyverse, tidymodels, embed (note we will need to install embed).

I’m assuming you’ve used the tidyverse before. If you have not used tidymodels or embed packages before, here’s a quick summary.

* tidymodels – next generation of packages that incorporate tidyverse principles into machine learning and modeling.
* embed – contains extra recipes steps to create “embeddings” (i.e., encoding predictors)

# install the 'embed' package  
#install.packages('embed')  
  
# load the needed packages  
library(tidyverse)

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --  
## v dplyr 1.1.4 v readr 2.1.5  
## v forcats 1.0.0 v stringr 1.5.1  
## v ggplot2 3.5.1 v tibble 3.2.1  
## v lubridate 1.9.3 v tidyr 1.3.1  
## v purrr 1.0.2   
## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()  
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(tidymodels)

## -- Attaching packages -------------------------------------- tidymodels 1.2.0 --  
## v broom 1.0.7 v rsample 1.2.1  
## v dials 1.3.0 v tune 1.2.1  
## v infer 1.0.7 v workflows 1.1.4  
## v modeldata 1.4.0 v workflowsets 1.1.0  
## v parsnip 1.2.1 v yardstick 1.3.1  
## v recipes 1.1.0   
## -- Conflicts ----------------------------------------- tidymodels\_conflicts() --  
## x scales::discard() masks purrr::discard()  
## x dplyr::filter() masks stats::filter()  
## x recipes::fixed() masks stringr::fixed()  
## x dplyr::lag() masks stats::lag()  
## x yardstick::spec() masks readr::spec()  
## x recipes::step() masks stats::step()  
## \* Dig deeper into tidy modeling with R at https://www.tmwr.org

library(embed)

## Warning: package 'embed' was built under R version 4.4.2

# set options to enlarge our plots  
options(repr.plot.width=12, repr.plot.height=16)

#### Load the Credit Data

The data was adapted from Kaggle’s “Credit score classification” data (thanks Rohan Paris!).

We’ll load it using read\_csv() and take a glimpse of it.

# the credit score data is available here  
data\_url <- "https://assets.datacamp.com/production/repositories/6081/datasets/e02471e553bc28edddc1fe862666d36e04daed80/credit\_score.csv"  
  
# use read\_csv to load the data  
credit\_df <- read\_csv(data\_url)

## Rows: 18965 Columns: 23  
## -- Column specification --------------------------------------------------------  
## Delimiter: ","  
## chr (6): month, occupation, credit\_mix, payment\_of\_min\_amount, payment\_beha...  
## dbl (17): age, annual\_income, monthly\_inhand\_salary, num\_bank\_accounts, num\_...  
##   
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

# reorder the credit\_score factor levels  
credit\_df <- credit\_df %>%   
 mutate(credit\_score = factor(credit\_score, levels = c("Poor", "Standard", "Good")))  
  
# look at the available features  
glimpse(credit\_df)

## Rows: 18,965  
## Columns: 23  
## $ month <chr> "January", "July", "April", "January", "Febru~  
## $ age <dbl> 44, 19, 39, 43, 22, 52, 32, 45, 37, 22, 26, 2~  
## $ occupation <chr> "Doctor", "Doctor", "Manager", "Developer", "~  
## $ annual\_income <dbl> 32625.590, 31041.460, 25876.180, 15928.750, 1~  
## $ monthly\_inhand\_salary <dbl> 2922.7992, 2501.7883, 2293.3483, 1301.3958, 1~  
## $ num\_bank\_accounts <dbl> 0, 5, 4, 9, 7, 5, 7, 9, 1, 7, 5, 7, 9, 4, 4, ~  
## $ num\_credit\_card <dbl> 5, 2, 6, 9, 9, 5, 8, 8, 5, 7, 5, 5, 5, 5, 7, ~  
## $ interest\_rate <dbl> 8, 5, 7, 24, 18, 5, 26, 21, 10, 25, 3, 23, 32~  
## $ num\_of\_loan <dbl> 4, 1, 3, 4, 2, 4, 9, 2, 2, 7, 1, 3, 6, 1, 2, ~  
## $ delay\_from\_due\_date <dbl> 5, 5, 5, 26, 40, 23, 62, 18, 13, 51, 10, 15, ~  
## $ num\_of\_delayed\_payment <dbl> 10, 11, 15, 22, 17, 1, 18, 18, 3, 16, 7, 21, ~  
## $ changed\_credit\_limit <dbl> 2.55, 5.13, 10.29, 1.26, 6.09, 0.75, 22.13, 8~  
## $ num\_credit\_inquiries <dbl> 3, 0, 4, 12, 12, 1, 12, 13, 3, 8, 0, 9, 12, 1~  
## $ credit\_mix <chr> "\_", "Good", "\_", "Bad", "Bad", "Good", "Bad"~  
## $ outstanding\_debt <dbl> 177.90, 291.77, 71.54, 2240.56, 2063.45, 636.~  
## $ credit\_utilization\_ratio <dbl> 39.41268, 25.26184, 25.24044, 31.18439, 23.13~  
## $ payment\_of\_min\_amount <chr> "No", "NM", "NM", "Yes", "NM", "No", "NM", "Y~  
## $ total\_emi\_per\_month <dbl> 73.125008, 21.021180, 37.374350, 31.846679, 1~  
## $ amount\_invested\_monthly <dbl> 139.43374, 185.31083, 117.93599, 161.38822, 1~  
## $ payment\_behaviour <chr> "Low\_spent\_Small\_value\_payments", "Low\_spent\_~  
## $ monthly\_balance <dbl> 369.7212, 333.8468, 354.0245, 226.9047, 343.9~  
## $ credit\_history\_months <dbl> 344, 398, 257, 145, 161, 199, 31, 121, 244, 1~  
## $ credit\_score <fct> Standard, Good, Standard, Poor, Poor, Poor, S~

The data’s dimensionality is just its number of columns. credit\_df has 23 dimensions, or features – one target variable (credit\_score) and 22 predictor variables.

The target variable – credit\_score – is categorical and has three levels: Poor, Standard, and Good. So, from a machine learning perspective we’ll be dealing with a classification problem.

Our core objective is to understand what differentiates consumers with poor, standard, and good credit scores. In short, we want to explain why consumers’ credit scores differ. Along the way, we’ll learn about UMAP (feature extraction algorithm) and the tidymodels framework.

### Exploration

Let’s visually explore credit\_df a little and see if we can understand why consumers have different credit scores.

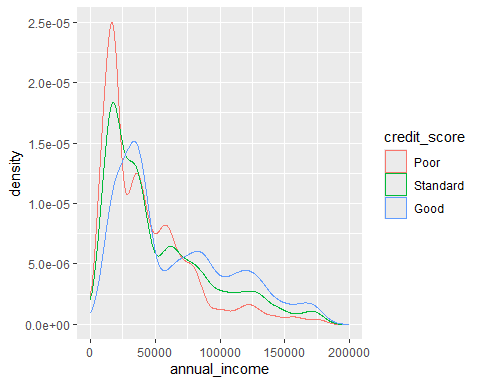
NOTE:: As humans we can’t visualize high-dimensional data – we are limited to about three dimensionals (maybe four, if you add animation to capture time).

What differentiates consumer credit scores? Let’s generate a few plots to see if we can discover a few predictors that do a good job of separating the credit scores.

### Annual income density plot

Let’s start by plotting the distribution of annual income for each of the three credit score levels.

# plot annual\_income distribution for each credit score level  
credit\_df %>%   
 ggplot(aes(x = annual\_income, color = credit\_score)) +  
 geom\_density() +  
 xlim(0, 200000)

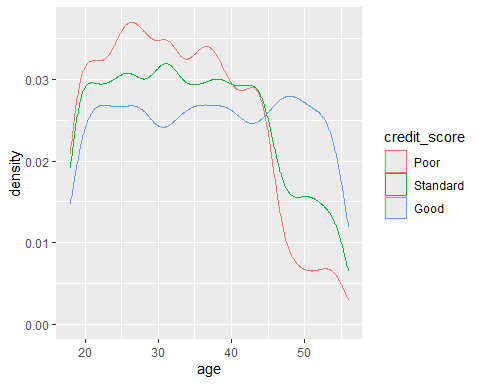


Takeaway: Those with lower annual income tend to have poorer credit scores. That means that annual income contains information that helps us determine credit score.

### Age density plot

Let’s explore the age of consumers by creating a density plot of age for each of the credit score levels.

# plot age distribution for each credit\_score level  
credit\_df %>%   
 ggplot(aes(x = age, color = credit\_score)) +  
 geom\_density()



Takeaway: Older consumers tend to have better credit score. In other words, age also contains some information that is useful for determining credit\_score.

### Delay from due date vs. credit history months

* Delay from due date = the average number of days late on payment
* Credit history months = the number of months of credit history the consumer has on record
* Let’s explore both of these features using a scatterplot that separates the credit score levels by color.

# plot delay\_from\_due\_date vs credit\_history\_months   
credit\_df %>%   
 ggplot(aes(x = delay\_from\_due\_date, y = credit\_history\_months , color = credit\_score)) +  
 geom\_jitter(alpha = 0.4)



Note that geom\_jitter() (instead of geom\_point()) prevents the data points from overlapping. This helps us to better see the colors of the three credit levels.

Takeaway: Both delay\_from\_due\_date and credit\_history\_months contain information about credit\_score.

### Credit utilization ratio vs number of credit cards

credit utilization = the amount of their credit limit the consumer has used

Like above, let’s explore credit utilization ratio and the number of credit cards a consumer has with a scatterplot.

# plot credit\_utilization\_ratio vs. num\_credit\_card  
credit\_df %>%   
 ggplot(aes(x = credit\_utilization\_ratio, y = num\_credit\_card, color = credit\_score)) +  
 geom\_jitter(alpha = 0.4) +  
 ylim(0, 10)



The number of credit cards separates the consumers’ credit levels pretty well. Not surprisingly, those with more credit cards tend to have poorer credit. So, num\_credit\_card has valuable information about a consumer’s credit score.

### Conclusion

It would take a long time to visit every combination of the predictors to determine which predictors help distinguish information about consumers’ credit levels. In this process, we’d discover that some predictors contain little to no information consumer’s credit levels. Dropping these features would simplify the data without significantly reducing the information in the data.

Dimensionality reduction is all about eliminating the useless noise and honing in on the “signal” – the useful information that helps us accomplish our goal in predicting the target variable.

#### Enter UMAP!

#### UMAP

UMAP is a feature extraction technique. We will use it to reduce our 22 predicting features (dimensions) down to 2 dimensions, while still keeping as much of the original information as possible. We’ll later demonstrate how much information was retained by fitting a decision tree model to predict credit score level on both the raw data and the UMAP tranformed data and observer the performance difference.

### Create the tidymodels recipe

A recipe in tidymodels is just like a cooking recipe – it’s a series of steps you apply to the data to get a predictable, repeatable desired outcome.

##### Our UMAP recipe will have two steps:

Normalize the data. We do this so that the unequal ranges of the features will not have unequal leverage in determining their importance in determing credit score level. Apply UMAP In the code below, we create the recipe object and then add these steps to it.

#### Here are some important things to notice:

We apply step\_normalize() only to the numeric predictors – we can’t normalize categorical variables and the target variable is categorical. We apply step\_umap() to all the predictors, but not the target variable because we need to keep that separate to train the model – UMAP can handle both continuous and categorical variables. We specify target variable with the outcome argument – outcome = vars(credit\_score). We specify that we want to reduce the data to two dimensionals with num\_comp in step\_umap().

# create the UMAP recipe with the normalize and UMAP steps  
umap\_recipe <- recipe(credit\_score ~ ., data = credit\_df) %>%   
 step\_normalize(all\_numeric\_predictors()) %>%   
 step\_umap(all\_predictors(), outcome = vars(credit\_score), num\_comp = 2)

###Train the recipe; and extract and plote the data

Now, let’s apply the recipe and get the transformed data so we can plot it.

The prep() function “trains” the recipe on the data. In other words, it applies UMAP to the data.

Then we use juice() to extract the (UMAP) transformed data from the recipe, which we will plot.

Before we plot the data, take a glimpse of it. Notice the two UMAP dimensions – data reduction complete!

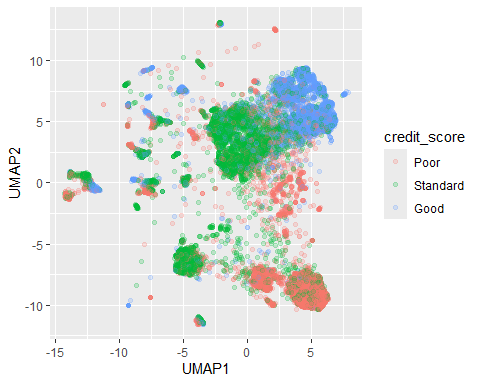
# take a look at the UMAP transformed data  
glimpse(umap\_credit\_df)

## Rows: 18,965  
## Columns: 3  
## $ credit\_score <fct> Standard, Good, Standard, Poor, Poor, Poor, Standard, Poo~  
## $ UMAP1 <dbl> 1.4454476, 6.4349661, -0.3442656, 4.5719490, 5.0182972, -~  
## $ UMAP2 <dbl> 2.4997737, 6.5436707, 3.0874496, -8.4260683, -9.1174746, ~

#### Plot the UMAP dimensions

Ultimately, UMAP is most useful for visualizing high-dimensional data. Here are some nice examples. So, let’s plot those two dimensions and color code the credit levels, like before, and see how well the UMAP transformation separates the credit score levels.

# Create a scatterplot of the UMAP dimensions, coloring credit score level  
umap\_credit\_df %>%  
 ggplot(aes(x = UMAP1, y = UMAP2, color = credit\_score)) +  
 geom\_point(alpha = 0.2)



UMAP does a nice job separating out the credit scores, right?

But you might be wondering, “How did it separate the credit scores?” In other words, what does UMAP1 and UMAP2 mean? That’s a downside to feature extraction – the extracted dimensions are often difficult to interpret.

### Building UMAP into a Decision Tree Model

Now, let’s turn our attention to integrating dimensionality reduction into a tidymodels workflow. In this training, we do this to illustrate the concept of information and that UMAP (like other feature extraction techniques), foundationally, seeks to retain information in the data while still it.

Let’s build a classification model with and without UMAP and compare their performance. The idea is that if UMAP cuts too much information, the classification model’s performance will take a big hit.

### Prepare the test and train datasets

Now, let’s create the train and test sets. tidymodels (specifically, the rsample package) makes this easy.

initial\_split() defaults to 75% for the training set and 25% for the testing set. We’ll accept those defaults.

Also, notice we set the random number generator seed to make this example reproducible.

# set seed to make this reproducible  
set.seed(3)  
  
# Initialize the data split  
credit\_split <- initial\_split(credit\_df)  
  
# Extract the train dataset  
train <- training(credit\_split)  
  
# Extract the test dataset  
test <- testing(credit\_split)

### Build a tidymodels workflow without UMAP

Now, it’s time to build the base model - the model from the original credit\_df data, without using UMAP.

A tidymodels workflow allows us to bundle recipe code with model building code – it’s a pipeline for pre-processing, modeling, and post-processing.

So, let’s create the recipe and the model and then bundle them into a worflow object.

### Create the recipe

In this case, it’s an empty recipe – we don’t want to modify the original credit\_df data. Notice that we pass the recipe object a formula – this basically says, “Use credit\_score as the target variable”.

# create an empty recipe object with credit\_score as the target variable  
dt\_recipe <- recipe(credit\_score ~ ., data = train)

### Create the decision tree model

The parnsip package in tidymodels provides a simple-to-use interface for creating models. Let’s instantiate a decision tree model for classification.

# create a decision tree model for classification  
dt\_model <- decision\_tree(mode = "classification")

Notice we specify mode = “classification” because many decision tree algorithms (e.g., random forest, CART, ID3) can perform regression.

NOTE: decision\_tree() abstracts away model implementation details. You can specifiy the implementation you use with the engine argument.

*Add the recipe and model to the workflow*

# create a workflow and add the recipe and model to it  
dt\_workflow <- workflow() %>%   
 add\_recipe(dt\_recipe) %>%   
 add\_model(dt\_model)

Train the model To be precise, we are fitting the recipe and the model on the training data set when we call this code

# train the model (that is, fit the workflow to the train data)  
dt\_fit <- dt\_workflow %>%   
 fit(data = train)

Predict the test data

# predict the test data with the trained model  
predict\_df <- test %>%   
 bind\_cols(predict = predict(dt\_fit, test))

#### Evaluate the model’s test predictions

The yardstick package in tidymodels provides an interface to evaluate models with a variety of metrics. Here we will use the F-measure, which combines the model recall and precision.

# Calculate the F-measure of the test predictions  
f\_meas(predict\_df, credit\_score, .pred\_class)

## # A tibble: 1 x 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 f\_meas macro 0.622

And that will serve has our benchmark comparison for UMAP. Remember, we are just getting a feel for how much information UMAP retained as it reduced the data to two dimensions.

#### Build a tidymodel workflow with UMAP

Now, let’s repeat the above process. The only difference is the recipe.

We’ll add the same two steps we used above when we plotted the UMAP data – step\_normalize() and step\_umap() – to preprocess the data before building the decision tree model.

NOTE: This decision tree model will only be trained on two predictors – UMAP1 and UMAP2; whereas, the prior model was trained on 22 predictors.

# create the UMAP recipe  
dt\_recipe <- recipe(credit\_score ~ ., data = train) %>%   
 step\_normalize(all\_numeric\_predictors()) %>%   
 step\_umap(all\_numeric\_predictors(), outcome = vars(credit\_score), num\_comp = 2)  
  
# create the decision tree model  
dt\_model <- decision\_tree(mode = "classification")  
  
# add the recipe and model to the workflow  
dt\_workflow <- workflow() %>%   
 add\_recipe(dt\_recipe) %>%   
 add\_model(dt\_model)  
  
# train the model  
dt\_fit <- dt\_workflow %>%   
 fit(data = train)  
  
# predict the test set  
predict\_df <- test %>%   
 bind\_cols(predict = predict(dt\_fit, test))  
  
# evaluate the model performance on the test set  
f\_meas(predict\_df, credit\_score, .pred\_class)

## # A tibble: 1 x 3  
## .metric .estimator .estimate  
## <chr> <chr> <dbl>  
## 1 f\_meas macro 0.561

With UMAP, the model’s performance (based on the F-measure) dropped to 0.548 (from 0.622). So, not surprisingly, we lost some information through the UMAP reduction. But for going from 22 to 2 dimensions, the model based on UMAP performs pretty well compared to the model based on the original credit\_df data. Depending on the context, this performance reduction may or may not be acceptable.