Why transform existing features?

FEATURE ENGINEERING IN R

Jorge Zazueta

Research Professor. Head of the Modeling Group at the School of Economics, UASLP

Making your model's life easier

We can improve the performance of our machine-learning model by making the data more manageable.

```
glimpse(loans_num)
```

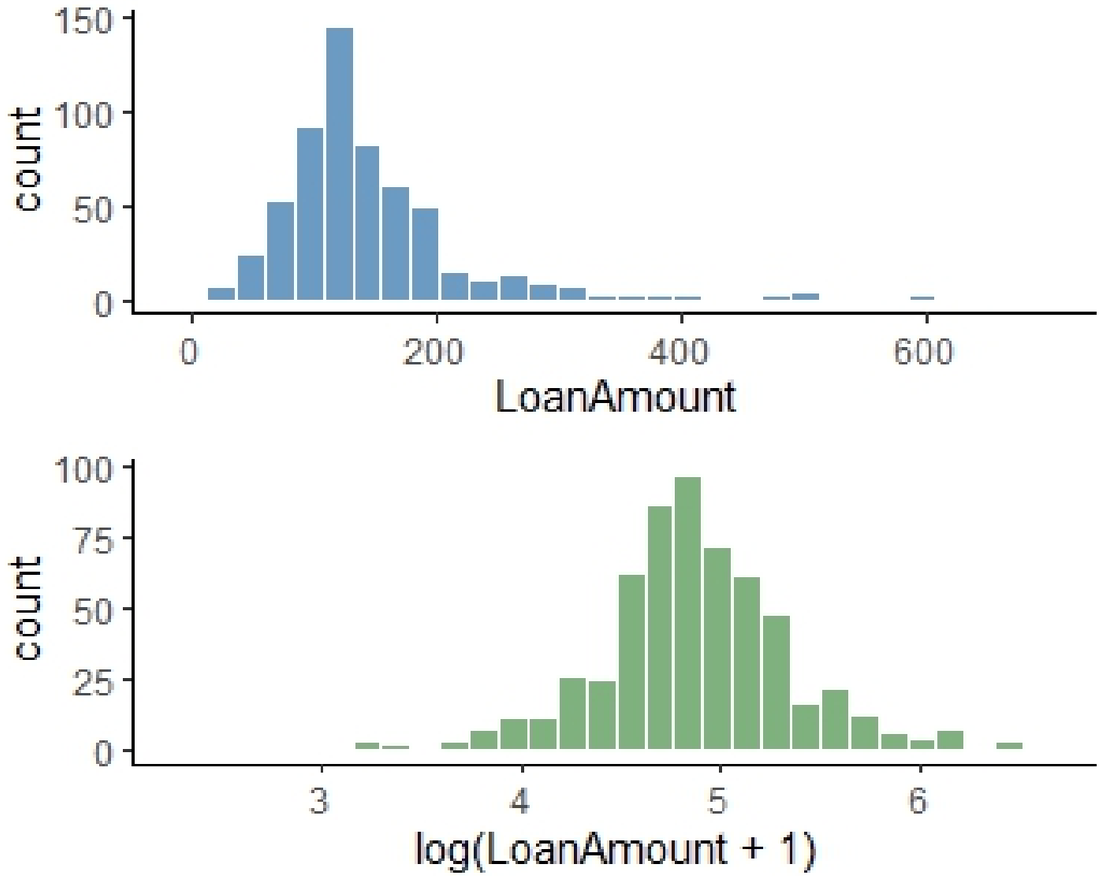
```
Rows: 614
Columns: 6
$ Loan_Status       <fct> Y, N, Y, Y, Y, Y, Y, N, Y, N, Y, Y, Y, N...
$ ApplicantIncome   <dbl> 5849, 4583, 3000, 2583, 6000, 5417, 233...
$ CoapplicantIncome <dbl> 0, 1508, 0, 2358, 0, 4196, 1516, 2504, 1...
$ LoanAmount        <dbl> NA, 128, 66, 120, 141, 267, 95, 158, 168...
$ Loan_Amount_Term  <dbl> 360, 360, 360, 360, 360, 360, 360, 360, ...
$ Credit_History    <fct> 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1...
```

Log transformation

log-transform numerical features to:

- Handle skewed data
- Reduce the impact of outliers
- Convert multiplicative relations into additive
- Make the data more suitable for modeling
- Works only for positive values

***log*-transformed loan amount data**

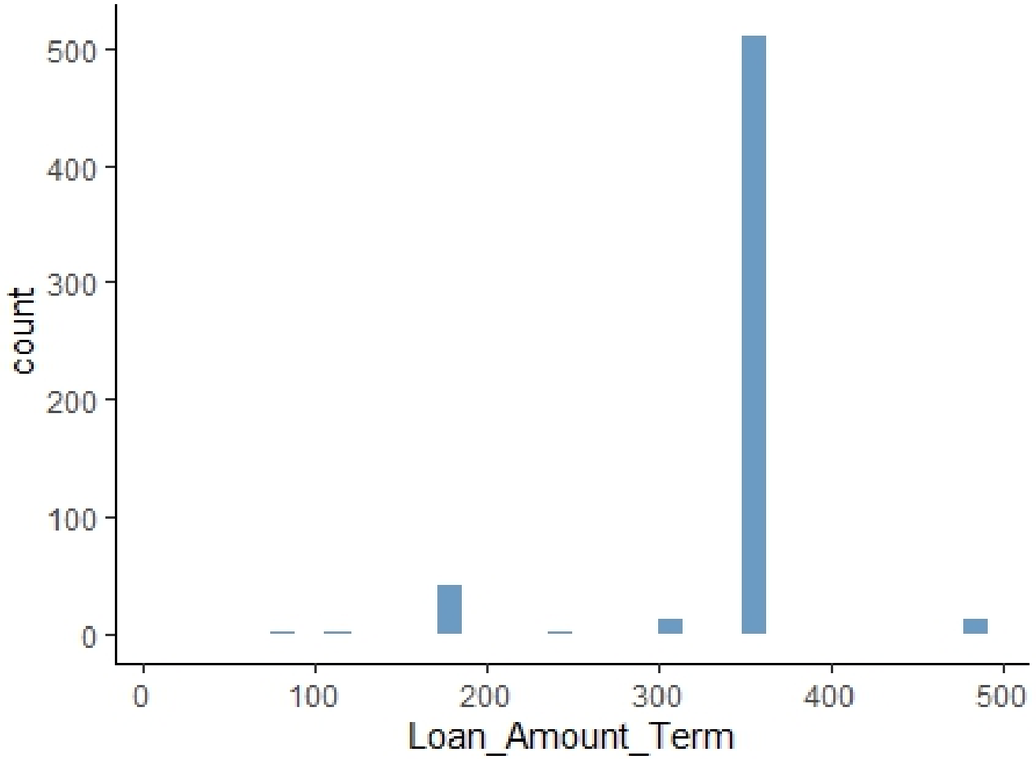


Normalization

Normalize or scale numerical features to:

- Prevent one feature from dominating the others
- Ease interpretation because it gives a comparable magnitude and
- Make the data more suitable for modeling

e.g., loan amount term values shown vary significantly

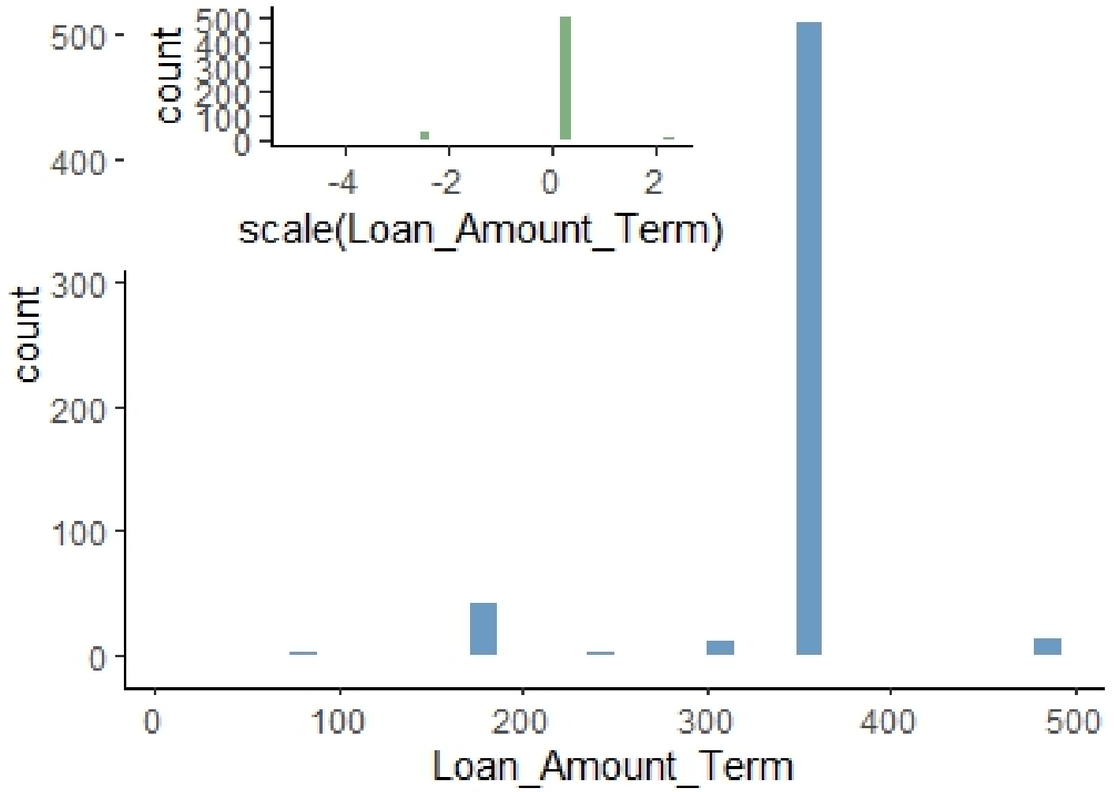


Normalization

Normalize or scale numerical features to:

- Prevent one feature from dominating the others
- Help handle outliers and
- Missing data
- Make the data more suitable for modeling

Normalized values preserve distribution, but contain variation.



Defining the model and the recipe

We can now declare a logistic regression model and add a recipe to impute, normalize and log-transform the relevant features.

```
lr_model <- logistic_reg()

lr_recipe <-
  recipe(Loan_Status ~.,
         data = train) %>%
  step_impute_knn(
    all_numeric_predictors())%>%
  step_normalize(Loan_Amount_Term) %>%
  step_log(all_numeric_predictors(),
           -Loan_Amount_Term, offset = 1)
```

Printing the recipe object shows a summary of the steps applied.

```
lr_recipe
```

```
Recipe

Inputs:

      role #variables
   outcome           1
 predictor           5

Operations:

K-nearest neighbor imputation for all_numeric_predictors()
Centering and scaling for Loan_Amount_Term
Log transformation on all_numeric_predictors(),-Loan_Amount_Term
```

Measuring performance efficiently

We define a set of metrics, `roc_auc`, `accuracy` and `sens` to assess the fit workflow object `lr_fit`.

```
class_evaluate <- metric_set(
  roc_auc, accuracy, sens)
```

And run it as you would any function.

```
lr_aug %>%
  class_evaluate(
    truth = Loan_Status,
    estimate = .pred_class,
    .pred_Y)
```

Customized set of metrics

```
# A tibble: 3 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.813
2 sens     binary         0.467
3 roc_auc  binary         0.288
```

Let's practice!

FEATURE ENGINEERING IN R

Common feature transformations

FEATURE ENGINEERING IN R

Jorge Zazueta

Research Professor and Head of the Modeling Group at the School of Economics, UASLP

Two families of transformations

Box-Cox

- Used to transform non-normal variable closer to normal
- As a family, it includes inverse, log, square and cubic roots as special cases
- Works for strictly positive values

$$\varphi(y,\lambda) = \begin{cases} \dfrac{y^{\lambda}-1}{y} & \lambda \neq 0, y > 0 \\ \log y & \lambda = 0, y > 0 \end{cases}$$

Yeo-Johnson

- Similar properties as Box-Cox
- Can handle zero and negative values
- For positive y is the same as Box-Cox of $y + 1$

$$\varphi(y,\lambda) = \begin{cases} \dfrac{(y+1)^{\lambda}-1}{\lambda} & \lambda \neq 0, y \geq 0 \\ \log(y+1) & \lambda = 0, y \geq 0 \\ -\dfrac{\left[(-y+1)^{2-\lambda}-1\right]}{2-\lambda} & \lambda \neq 2, y < 0 \\ -\log(-y+1) & \lambda = 2, y < 0 \end{cases}$$

The loans_num dataset

```
glimpse(loans_num)
```

```
Rows: 480
Columns: 6
$ Loan_Status       <fct> N, Y, Y, Y, Y, Y, N, Y, N, Y, Y, N, Y, Y, N...
$ ApplicantIncome   <dbl> 4583, 3000, 2583, 6000, 5417, 2333, 3036, 4...
$ CoapplicantIncome <dbl> 1508, 0, 2358, 0, 4196, 1516, 2504, 1526, 1...
$ LoanAmount        <dbl> 128, 66, 120, 141, 267, 95, 158, 168, 349, ...
$ Loan_Amount_Term  <dbl> 360, 360, 360, 360, 360, 360, 360, 360, 360...
$ Credit_History    <fct> 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0...
```

Applying transformations

Plain recipe

```
lr_recipe_plain <- # Define recipe
  recipe(Loan_Status ~., data = train)
lr_workflow_plain <- # Bundle workflows
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_plain)
lr_fit_plain <- # fit and augment
  lr_workflow_plain %>%
  fit(train)
```

Assess performance

```
lr_aug_plain %>% # Assess
  class_evaluate(truth = Loan_Status,
                 estimate = .pred_class,
                 .pred_N)
```

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.817
2 roc_auc  binary         0.641
```

Applying transformations

Box-Cox recipe

```
lr_recipe_BC <- # Define recipe
  recipe(Loan_Status ~., data = train) %>%
  step_BoxCox(all_numeric())
lr_workflow_BC <- # Bundle workflows
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_BC)
lr_fit_BC <- # fit and augment
  lr_workflow_BC %>%
  fit(train)
```

Warning Message

Box-Cox is unable to process non-positive values

```
Warning messages:
1: Non-positive values in selected
variable.
2: No Box-Cox transformation could be
estimated for: `CoapplicantIncome`
```

Applying transformations

Box-Cox recipe (take two)

Now, let's deselect `CoappliantIncome` to avoid the warning.

```
lr_recipe_BC <- # Define recipe
  recipe(Loan_Status ~., data = train) %>%
  step_BoxCox(all_numeric(),
              -CoapplicantIncome)
lr_workflow_BC <- # Bundle workflows
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_BC)
lr_fit_BC <- # fit and augment
  lr_workflow_BC %>%
  fit(train)
```

Assess performance

```
lr_aug_BC %>% # Assess
  class_evaluate(truth = Loan_Status,
                 estimate = .pred_class,
                 .pred_N)
```

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.817
2 roc_auc  binary         0.599
```

Applying transformations

Yeo-Johnson recipe

```
lr_recipe_YJ <- # Define recipe
  recipe(Loan_Status ~., data = train) %>%
  step_YeoJohnson(all_numeric())
lr_workflow_YJ <- # Bundle workflows
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_YJ)
lr_fit_YJ <- # fit and augment
  lr_workflow_YJ %>%
  fit(train)
```

Assess performance

```
lr_aug_YJ %>% # Assess
  class_evaluate(truth = Loan_Status,
                 estimate = .pred_class,
                 .pred_N)
```

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.817
2 roc_auc  binary         0.700
```

Let's practice!

FEATURE ENGINEERING IN R

Advanced transformations

FEATURE ENGINEERING IN R

Jorge Zazueta

Research Professor and Head of the Modeling Group at the School of Economics, UASLP

The data

```
Rows: 480
Columns: 6
$ Loan_Status       <fct> N, Y, Y, Y, Y, Y, N, Y, N, Y, Y, N, Y, Y, N, N, ...
$ ApplicantIncome   <dbl> 4583, 3000, 2583, 6000, 5417, 2333, 3036, 4006, ...
$ CoapplicantIncome <dbl> 1508, 0, 2358, 0, 4196, 1516, 2504, 1526, 10968,...
$ LoanAmount        <dbl> 128, 66, 120, 141, 267, 95, 158, 168, 349, 70, 2...
$ Loan_Amount_Term  <dbl> 360, 360, 360, 360, 360, 360, 360, 360, 360, 360...
$ Credit_History    <fct> 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, ...
```

Predict with plain workflow

Configure the recipe and set the workflow

```
lr_recipe_plain <-
  recipe(Loan_Status ~., data = train)
lr_workflow_poly <-
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_plain)
```

Fit and assess the workflow

```
lr_fit_plain <-
  lr_workflow_plain %>% fit(train)
lr_aug_plain <-
  lr_fit_plain %>% augment(test)
lr_aug_plain %>%
  class_evaluate(truth = Loan_Status
```

Plain recipe results.

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.75
2 roc_auc  binary         0.595
```

The step_poly() function

`step_poly()` implements a polynomial expansion to one or more variables and passes it to our model.

```
lr_recipe_poly <-
  recipe(Loan_Status ~., data = train) %>%
  step_poly(all_numeric_predictors())
```

```
lr_workflow_poly <-
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_poly)
```

Results with `step_poly()`

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.75
2 roc_auc  binary         0.703
```

The step_percentile() function

`step_percentile()` determines the empirical distribution of a variable based on the training set and converts all values to percentiles.

```
lr_recipe_perc <-
  recipe(Loan_Status ~., data = train) %>%
  step_percentile(all_numeric_predictors())
```

```
lr_workflow_perc <-
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_perc)
```

Results with `step_percentile()`

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.769
2 roc_auc  binary         0.677
```

Let's practice!

FEATURE ENGINEERING IN R