Week 2 AHA: Gradient Descent

2024-02-06

Install packages

```
# Install {latentFactoR} (to categorize data)
remotes::install_github("AlexChristensen/latentFactoR")
```

Load packages

```
# Load {latentFactoR} (to categorize data)
library(latentFactoR)
```

Load source function

```
# Source the gradient descent function
source("../materials/source/glm_gradient_descent.R")
```

Set seed

```
# Set seed to ensure reproducible results
set.seed(42)
```

Generate data and perform linear regression

```
# Generate data
X <- rnorm(1000, mean = 0, sd = 0.10)
Y <- X + rnorm(1000)

# OLS
lm(Y ~ X)
```

```

Call:
lm(formula = Y ~ X)

Coefficients:
(Intercept)            X
  -0.005064     1.098180
```

Get parameters to match OLS solution

```
# Arguments
# X = predictors (intercept not needed)
# Y = outcome
# learning_rate = steepness of descent (default = 0.01)
# max_iter = maximum number of iterations (default = 1000)
```

```r
# activation = linear or sigmoid (default = "linear")

# Adjust the parameters of the gradient descent
# to get near the OLS output
glm_gradient_descent(
  X, Y, learning_rate = 0.01,
  max_iter = 1000000, activation = "linear"
)$best_result
```

```
       iteration             cost           beta_0           beta_1
167505.000000000    971.247174387     -0.005064449      1.098180299
```

```r
# Perform OLS
coef(lm(Y ~ X))
```

```
 (Intercept)            X
-0.005064449  1.098180353
```

If sticking with the default `learning_rate = 0.01`, then the `max_iter` parameter will need to be much, much larger. To get nearest to the lowest OLS solution (with the lowest cost), `max_iter` need to be increased to 1 million (`1000000`). The lowest cost occurred at iteration `167505`.

Conversely, increasing the `learning_rate` can get us to the proper solution faster:

```r
# Adjust the parameters of the gradient descent to get near the OLS output
glm_gradient_descent(
  X, Y, learning_rate = 0.1,
  max_iter = 100000, activation = "linear"
)$best_result
```

```
      iteration            cost          beta_0          beta_1
16799.000000000   971.247174387    -0.005064449     1.098180302
```

```r
# Perform OLS
coef(lm(Y ~ X))
```

```
 (Intercept)            X
-0.005064449  1.098180353
```

By increasing the `learning_rate` to `0.1` (or 10x greater learning), there are 10x fewer iterations (`16799`) needed to achieve the same results.

Categorize data for logistic regression

```r
# Convert Y to binary
Y <- categorize(Y, 2) - 1
```

Get parameters to match IRLS solution

```r
# Adjust the parameters of the gradient descent
# to get near the logistic regression output
glm_gradient_descent(
  X, Y, learning_rate = 0.1,
  max_iter = 100000, activation = "sigmoid"
)$best_result
```

```
     iteration           cost         beta_0         beta_1
62754.00000000   691.10478841    -0.04106398     1.20138406
```

```r
# Logistic regression
coef(glm(Y ~ X, family = "binomial"))
```

```
(Intercept)           X
-0.04106398  1.20138426
```

Doing the same adjustments on both `learning_rate` and `max_iter` can lead to equivalent results as logistic regression.

Goal of Activity

The purpose of this activity was to have you familiarize yourself with the way many models work in data science. Algorithmic approaches to modeling rely less on analytic solutions that are common in the social sciences – that is, data modeling.

Quick sidebar: In traditional statistics and the social sciences (data modeling), data are molded to a model. When using a linear model, for example, you get out a linear relationship whether or not the relationship between your $\mathbf{X}$ *and subsequent predictions are linearly related to your* $\mathbf{Y}$*. The linear model models linear relationships. The extent to which your data and subsequent predictions can be "molded" into a linear relationship determines how well the model "fits" the data (often, indicated by* R^2*).*

Algorithmic modeling uses non-parametric, unconstrained models that seek to maximally identify the relationship between $\mathbf{X}$ and $\mathbf{Y}$ – whatever that relationship might be. These algorithms are not a one-size-fits-all style model like more traditional statistics. Rather than data being molded to the model, the model is molded to the data. This flexibility comes with an extra requirement: Algorithmic models come with hyperparameters (e.g., `learning_rate`) that require adjustment with each and every dataset. The optimal parameters are not known because the underlying structure of the relationship between $\mathbf{X}$ and $\mathbf{Y}$ is assumed to be not known. Compare this notion to linear regression where the underlying structure of the relationship is assumed to be linear.