CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2016
Doc 13 Generalized Linear Models
Oct 6, 2016

Generalized Linear Models

Generalized linear regression to handle other cases (distributions)

- Linear regression
- Logistic regression
- Probit regression
- Poisson regression
- ...

Logistic regression

- Finite possible outcomes

Generalized Linear Models

Julia GLM package use the function

glm(formula, data, family, link)

formula	Y~X+Z
data	Dataframe
family (predictor)	Bernoulli(), Binomial(), Gamma(), Normal(), or Poisson()
link	Function linking predictor and mean of distribution

Family	Standard Link
Bernoulli	LogitLink
Binomial	LogitLink
Gamma	InverseLink
Normal	IdentityLink
Poisson	LogLink

Julia glm, lm and fit

lm and glm are convenience methods for fit

lm(Y~X+Z,a_data_frame) calls
    fit(LinearModel, Y~X+Z,a_data_frame)

glm(formula, data, family, link) calls
    fit(GeneralizedLinearModel, formula, data, family, link)

Categorical Variable

Variable takes on one of limited, usually fixed possible values

- Blood type of a person
- Political party a person will vote for
- State that one lives in

If only two possible values normally encoded as 1 & 0

Categorical variables need to handled differently in regression model

Logistic (Logit) Regression or Logit Model

Regression model where the dependent variable is categorical

Used to predict

- If a patient has a disease based on age, sex, blood tests, etc
- If a voter will vote Democratic or Republican
- If a product will fail

Logit Model in Julia

Use

Family	Binomial()
Link	LogitLink()

```
glm(formula, dataframe, Binomial(), LogitLink())
```

Hours Studied & Passing Exam

When only two outcomes encoded 1 & 0

Build model to predict given study time the probability of passing

Pass	Hours

Example from https://en.wikipedia.org/wiki/Logistic_regression

Generating the Model

```
using DataFrames
using GLM
using Distributions
```

```
hours = [0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 1.75, 2.00, 2.25, 2.50, 2.75, 3.00,
         3.25, 3.50, 4.00, 4.25, 4.50, 4.75]
pass = [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1,]

study_data = DataFrame(Hours= hours, Pass = pass)

study_model = glm(Pass~Hours, study_data, Binomial(), LogitLink())
show(study_model)
```

```
Formula: Pass ~ 1 + Hours

Coefficients:
             Estimate Std.Error  z value Pr(>|z|)
(Intercept)  -3.96352   1.78902 -2.21547   0.0267
Hours          1.4533  0.649233  2.23849   0.0252
```

Confidence Intervals

confint(study_model)

confint works on linear models too

-7.46994	-0.457101	Intercept
0.180829	2.72578	Hours

```
Formula: Pass ~ 1 + Hours

Coefficients:
             Estimate Std.Error  z value Pr(>|z|)
(Intercept)  -3.96352   1.78902 -2.21547   0.0267
Hours          1.4533  0.649233  2.23849   0.0252
```

Using Model to Predict

Not fitting data to a line
Fitting it to the logistic function

F(x)= 1/(1 + exp(Intercept + DependentVarEstimate*x)
= 1/(1 + exp(-3.96352 + 1.4533*x)

	Estimate	Std.Error	z value	Pr(>|z|)
(Intercept)	-3.96352	1.78902	-2.21547	0.0267
Hours	1.4533	0.649233	2.23849	0.0252

Generalizing the Function

probability(model, x) = 1/(1+exp(-(coef(model)[1]+coef(model)[2]*x)))

probability(study_model,4)

Hours Studied	How calculated	Probability of Passing
1		0.075
2		0.258
3		0.598
4	(study_model,4)	0.864

predict

Linear regression and Logistic regression are fitted to different equations

The Julia model knows which equation is to be used

GLM package function **predict** will fit the data

Hours Studied	How calculated	Probability of Passing
1		0.075
2		0.258
3		0.598
4		0.864

predict arguments

predict(study_model, [1.0 3.0])

predict computes

[1.0 3.0] * coef(study_model)

Then feeds the result into the proper fit (link) function

Since the first coefficient is the intercept the first value needs to be 1

Using DataFrames with predict

```
student = DataFrame(Hours = [3.0])

result_array = predict(study_model, student)

result_array[1] == 0.598
```

Works with linear models too

Generating Tables

```
students = DataFrame(Hours = [1.0, 2.0, 3.0, 4.0])

result_array = predict(study_model, students)
result_array
```

[0.0751, 0.258, 0.598, 0.864]

Second Example - Admission to Grad School

Data from http://www.ats.ucla.edu/stat/data/binary.csv

Analysis using R: http://www.ats.ucla.edu/stat/r/dae/logit.htm

Given the data build a model of admissions

admit - dependent variable, 1 = admit

gre, gpa

rank - ranking of school student attended, 1= highest rank, 4 lowest

Row	admit	gre	gpa	rank
1	0	380	3.61	3
2	1	660	3.67	3
3	1	800	4.0	1
4	1	640	3.19	4
5	0	520	2.93	4
6	1	760	3.0	2

Data was generated. Not real data on admissions from UCLA

Summary of the Data

admit		gre		gpa		rank	
Min	0.0	Min	220.0	Min	2.26	Min	1.0
1st Qu.	0.0	1st Qu.	520.0	1st Qu.	3.13	1st Qu.	2.0
Median	0.0	Median	580.0	Median	3.395	Median	2.0
Mean	0.3175	Mean	587.7	Mean	3.39	Mean	2.485
3rd Qu.	1.0	3rd Qu.	660.0	3rd Qu.	3.67	3rd Qu.	3.0
Max	1.0	Max	800.0	Max	4.0	Max	4.0

rank - Categorical Variable

rank ony has four values

So needs to be handled differently - Use PooledDataArray

Pooling Data

Categorical data in arrays can be space inefficient

Republican
Democrat
Democrat
Democrat
Republican
Republican

PooledDataArray

1 → Republican
2 → Democrat
2 → Democrat
2
1
1

Pooling DataFrame Columns

Use pool!(dataframe, [Columns_to_pool])

parties = [“Republican”, “Democrat”, “Democrat”, “Democrat”, “Republican”, “Republican”]

party_df = DataFrame(Party = parties)

pool!(party_df, [:Party])

Creating Admission Model

```
using DataFrames
using GLM
using Distributions

admit_data = readtable("admit_data.csv")

pool!(admit_data,[:rank])

admit_model = glm(admit~gre + gpa + rank, admit_data, Binomial(),LogitLink())
```

The Model

show(admit_model)

```
Formula: admit ~ 1 + gre + gpa + rank

Coefficients:
                Estimate  Std.Error  z value Pr(>|z|)
(Intercept)     -3.98998    1.13982 -3.50052   0.0005
gre           0.00226443 0.00109389  2.07007   0.0384
gpa             0.804037   0.331783  2.42338   0.0154
rank: 2        -0.675443    0.31648 -2.13423   0.0328
rank: 3          -1.3402   0.345284 -3.88146   0.0001
rank: 4         -1.55146   0.417804 -3.71337   0.0002
```

Confidence Intervals

confint(admit_model)

			Estimat
-6.22399	-1.75596	(Intercept)	-3.98998
0.000120448	0.0044084	gre	0.00226443
0.153755	1.45432	gpa	0.804037
-1.29573	-0.0551526	rank: 2	-0.675443
-2.01695	-0.663461	rank: 3	-1.3402
-2.37034	-0.732582	rank: 4	-1.55146

Using the Model

```
sample_student = DataFrame(gre=588,gpa=3.39,rank=2)
pool!(sample_student,[:rank])

result_array = predict(admit_model,sample_student)

result_array[1] = 0.352
```

Predicting Multiple Data

```
average_student = DataFrame(gre=fill(588,4), gpa=fill(3.39,4), rank=1:4)
pool!(average_student,[:rank])

show(head(average_student))
```

Row	gre	gpa	rank
1	588	3.39	1
2	588	3.39	2
3	588	3.39	3
4	588	3.39	4

Predicting Multiple Data

predict(admit_model,average_student)

Result

	Row	gre	gpa	rank
0.516791	1	588	3.39	1
0.352458	2	588	3.39	2
0.218742	3	588	3.39	3
0.184783	4	588	3.39	4