Classification

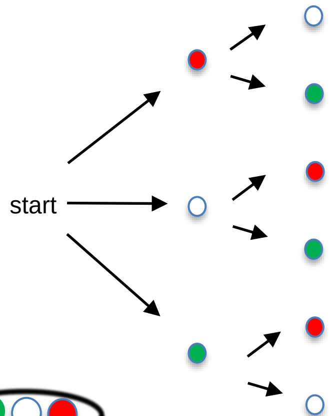
Primer: Conditional Probability and Bayes Theorem

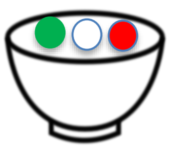
- Classification
  - Overview
  - Methods
    - Logistic Regression
    - Linear Discriminant Analysis
    - Naïve Bayes
    - Point Bayes
- Decision Trees
  - Overview
- Support Vector Machines

ISL Chapter 4

ISL Chapter 8

Conditional Probability and Independence

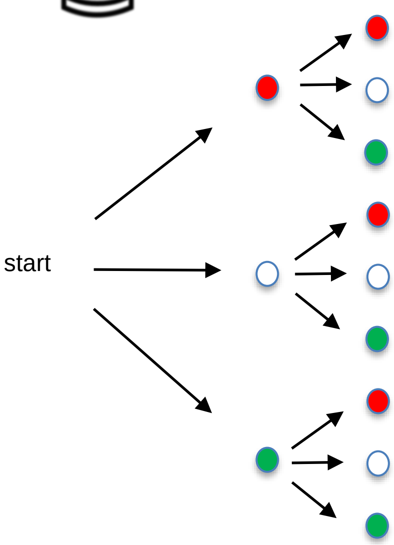




Two events are dependent if they do affect one another *(sampling without replacement)*

3 x 2 = 6 possible outcomes

$$\mathbb{P}(A \mid B) = \mathbb{P}(A) * \mathbb{P}(B|A)$$ 1/3 * 1/2 = 1/6



Two events are independent if they do not affect one another *(e.g., sampling with replacement)*

3 x 3 = 9 possible outcomes

$$\mathbb{P}(A \mid B) = \mathbb{P}(A) * \mathbb{P}(B)$$ 1/3 * 1/3 = 1/9

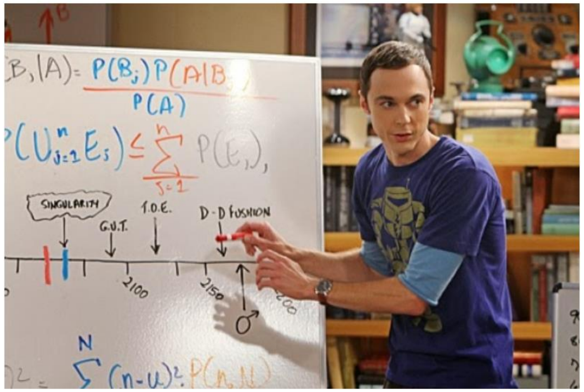
if A and B are independent events then the occurrence of A does not affect B, and $\mathbb{P}$ (B | A) becomes just $\mathbb{P}$(B).

Bayes Theorem

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(B \mid A)\mathbb{P}(A)}{\mathbb{P}(B)}$$

Reps in Territory / ***Reps at Quota***

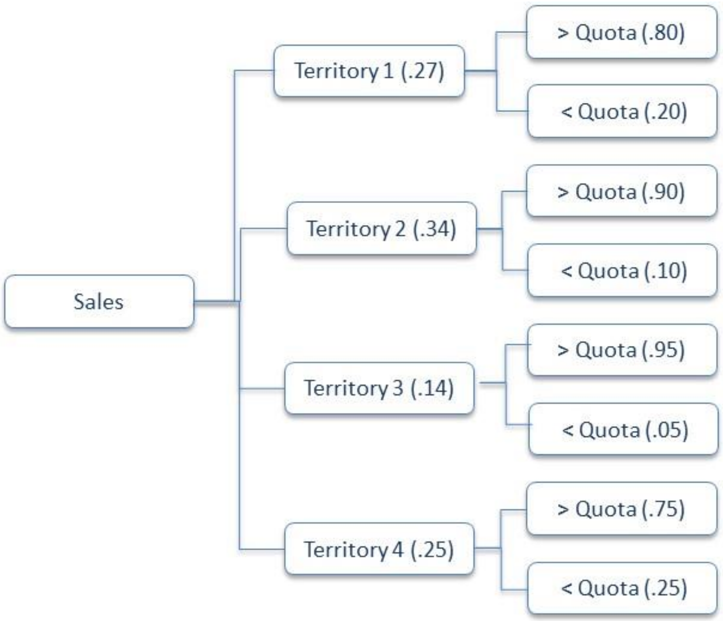
- Sales
  - Territory 1 (40)
    - > Quota (32)
    - < Quota (8)
  - Territory 2 (50)
    - > Quota (45)
    - < Quota (5)
  - Territory 3 (20)
    - > Quota (19)
    - < Quota (1)
  - Territory 4 (36)
    - > Quota (27)
    - < Quota (9)



Its easy to determine the probability of making quota, given a territory – e.g., if a rep works in territory 1, the probability of making quota is 80% (32/40).

But, what's the probability of a rep who makes quota, working in Territory 4? This is an "inverse" question, and the formula to find this was proposed by Thomas Bayes (1701-1761).

$$\mathbb{P}(T = 4 \mid Q = T) = \frac{\mathbb{P}(Q = T \mid T = 4)\ \mathbb{P}(T = 4)}{\mathbb{P}(Q = T)}$$



(converted to percentages)

First, What's the probability of making quota if rep is in Territory 4

$\mathbb{P}(A \mid B)$ = 75%

Now, *what's the probability of a rep who made quota being in Territory 4?* $\mathbb{P}(B \mid A)$

Bayesian approach:

$$\mathbb{P}(T = 4 \mid Q = T) = \frac{\mathbb{P}(Q = T \mid T = 4)\ \mathbb{P}(T = 4)}{\mathbb{P}(Q = T)}$$

26/36 *36/145*

$$= \frac{(.75 * .25)}{(.27 * .80) + (.34 * .90) + (.14 * .95) + (.75 * .25)} = .22$$

123/146

A tree structure can be easily set up in a spreadsheet (recommend this approach for homework 1)

Bayesian Update Table *(more flexible / efficient approach)*

Without likelihood data, our prior is the % of reps in each territory. This is an informed prior

This is the probability of observing quota data – the Likelihood.

This is the probability of observing H given the evidence – notice how the prior shifted up or down depending on the data.

Hypothesis	Prior	Likelihood	Bayes Numerator	Posterior
Territory 1	0.27	0.80	0.22	0.26
Territory 2	0.34	0.90	0.31	0.36
Territory 3	0.14	0.95	0.13	0.16
Territory 4	0.25	0.75	0.19	0.22
	1.00		0.84	1.00

The Posterior is the reallocation of credibility using the normalizing constant

The next time through, these posteriors become the prior (and if the data doesn't change, neither will the posteriors – prove it out). But what if some rep in territory 1 gets a "bluebird"? What's a better forecast for next year?

This is the numerator we just used

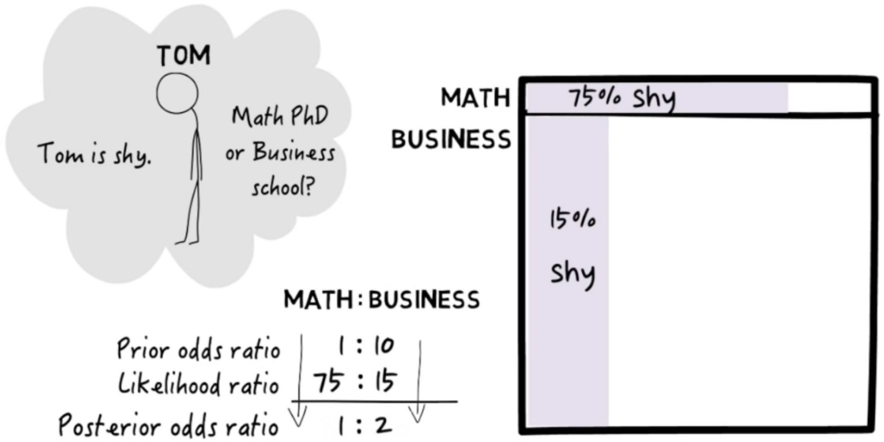
This is the denominator: P(Q=T)

$$\mathbb{P}(\text{Territory} \mid Q = T) = \frac{\mathbb{P}(Q=T)\ \mathbb{P}(\text{Territory} \mid Q=T)}{\mathbb{P}(Q=T)}$$

Another example:

Adapted from Julia Galef: [A visual guide to Bayesian thinking - YouTube]

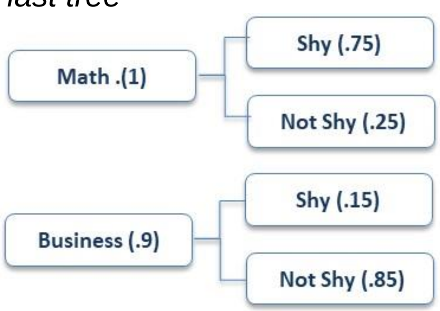
You meet a student who is shy. Is he more likely to be a math student or a business student?



$\mathbb{P}(\mathrm{H}) = 0.10$ *This is the* ***Prior*** *(estimate of the probability of Hypothesis B before the data A).*

$\mathbb{P}(\mathrm{D} \mid \mathrm{H}) = 0.75$ *This is the* ***Likelihood*** *(probability of observing D given H)* ***L is dynamic!***

One last tree



$$\mathbb{P}(\mathrm{D}) = (0.10 * .75) + (0.90 * 0.15) = 0.21$$

$$\text{Shy given Math PhD: } \mathbb{P}(\mathrm{D} \mid \mathrm{H}) = \frac{(0.10)*(0.75)}{0.21} = .36$$

$$\text{Shy given MBA: } \mathbb{P}(\mathrm{D} \mid \mathrm{H}) = \frac{(0.90)*(0.15)}{0.21} = .64$$

Note that the proportion $\propto$ of the posterior to the numerators is the same without the denominators, so the denominators (marginal) can be dropped for relative probability (ratios) and ***inference****. Restated: the posterior probability (what we're trying to measure) is proportional to its prior probability and the likelihood.*

Bayesian Update Table.

Hypothesis	Prior	Likelihood	Bayes Numerator	Posterior
Math	0.10	0.75	0.08	0.36
MBA	0.90	0.15	0.14	0.64
	1.00		**0.21**	**1.00**

The hard part of all this is defining the Bayes denominator (.21), the P(Data). We often don't know this, as the conditions can get complex and we many not have any estimates of the entire population *(the good news is – we can skip that if we know what the posterior looks like and we can estimate that by sampling)*.

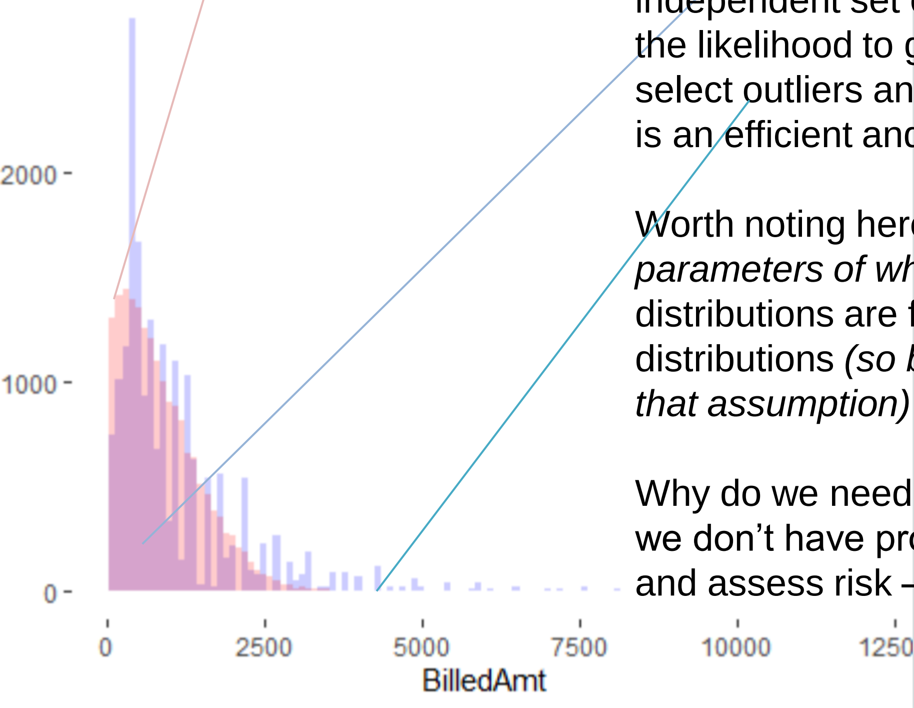
Application Scenario

You're auditing revenue transactions for 2020. You have both revenue transactions for 2019 and 2020 – 2019 has been audited. According to independent industry metrics, billing prices increased 20% from 2019-2020.

Posterior ***Likelihood*** ***Prior***

$$\mathbb{P}(\text{BilledAmt} > \text{x} \mid \text{YE} = 2020) \propto \mathbb{P}(\text{BilledAmt} > \text{x} \mid \text{YE} = 2019) * \mathbb{P}(\text{Theoretical Dist. based on Industry Metrics})$$

This approach starts with an independent and audited distribution of 2020 transactions. That's the likelihood. To that, we add a prior (another independent set of evidence). We combine (actually multiply) the prior * the likelihood to get the posterior. Once we have the posterior, we can select outliers and validate *(there's a statistical approach for that too)*. This is an efficient and credible approach.



Worth noting here that this is a skew normal distribution *(with 3 parameters of which do not include mean and standard deviation)* and SN distributions are far more common in transaction data than normal distributions *(so be very careful about sampling and inferences based on that assumption)*.

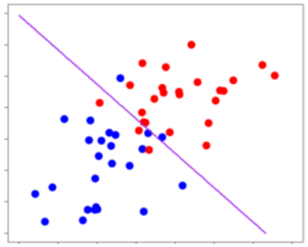
Why do we need to create a distribution? Because without a distribution, we don't have probabilities. Without probabilities, we can't make decisions and assess risk – we're just guessing.

Classification Methods

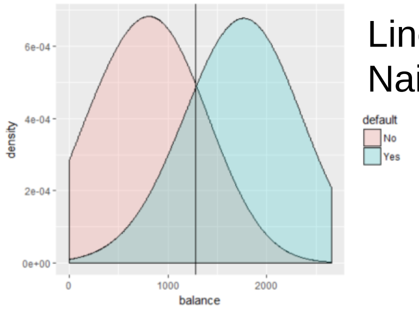
Classification is the problem of identifying to which of a set of categories *(sub-populations)* an observation belongs. Formally, given training set (x_i, y_i) for i=1…n, we want to create a classification model f that can determine the label y for x.

We'll survey a range of parametric and non-parametric algorithms:

Parametric

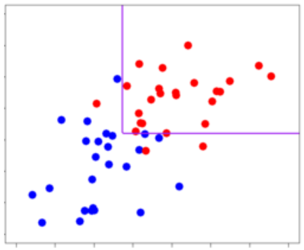


Logistic Regression

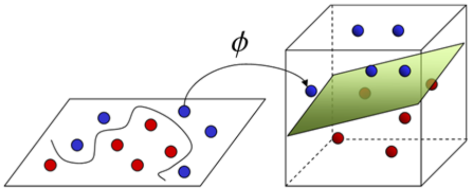


Linear Discriminant Analysis
Naive Bayes

Non-Parametric



Decision Trees



Support Vector Machines

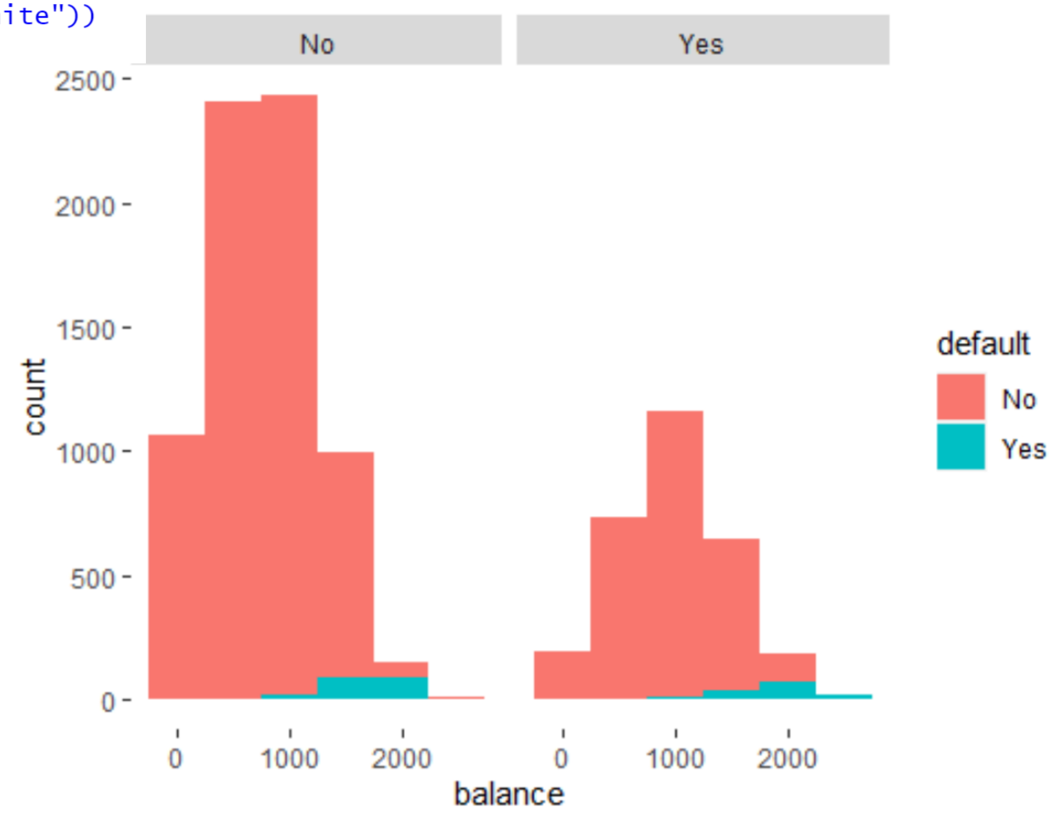
Logistic Regression

Credit Card Default Data

```
dfDefault <- Default

p <- ggplot(dfDefault, aes(balance, fill = default)) +
  geom_histogram(binwidth = 500) +
  facet_wrap(~student) +
  theme(panel.background = element_rect(fill = "white"))
p
```

We're interested in being able to determine whether an applicant will default.

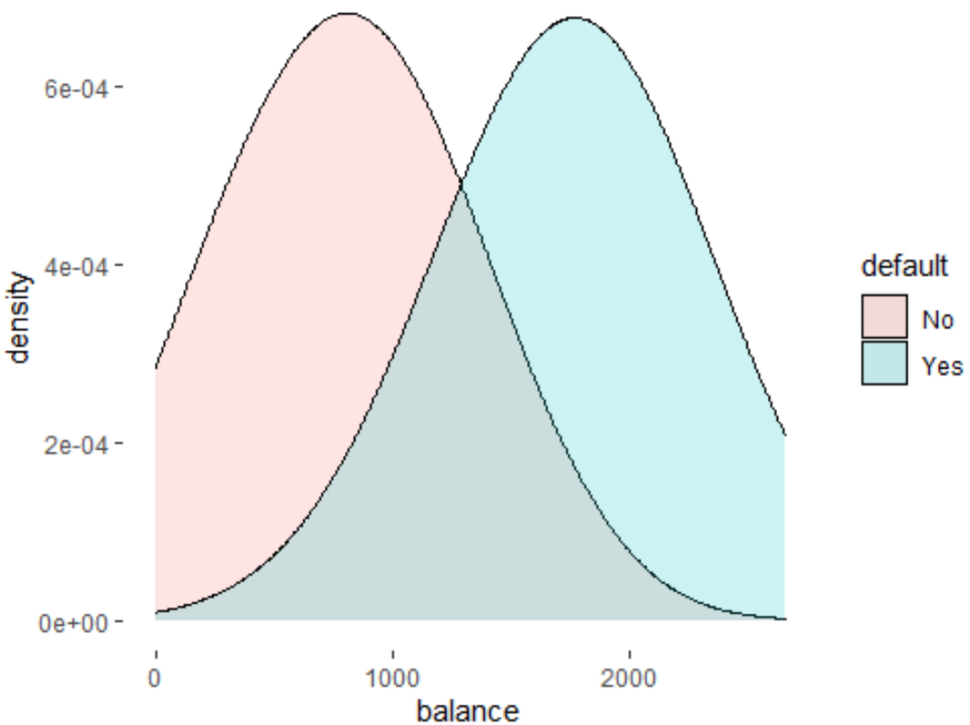
Note: For those going into Audit and Assurance, LR is the default tool for testing controls, which determines the extent of substantive testing (which costs money and irritates clients).



Looking at the distributions of default = No and default = Yes, you can see that there's a clear difference in means.

If you were just using the mean and increasing balance, at what point, does the probability of default become higher than No?

```
pl1 <- ggplot(dfDefault, aes(balance, fill = default)
  geom_density(alpha = 0.2, adjust = 5 ) +
  theme(panel.background = element_rect(fill = "whit
pl1
```



Logistic Regression

The logistic model starts with a linear model:

$y = \beta_0 + \beta_1 X$ *where* P(y=1,0 | X)

Since we now want to model P(y = 1 | X), and we know that probability must be 0 > P(y) > 1. So, we transform the equation to exponential form *(so it's always > 0)* and to a reciprocal *(so it's always < 1):*

$$P(y) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} \propto \log\left(\frac{P(x)}{1-P(x)}\right)$$

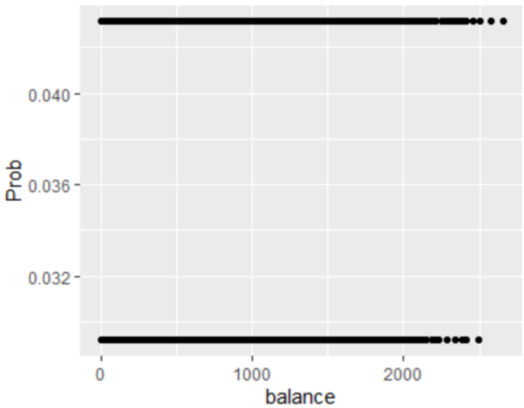
Modeling one categorical variable

```
dfDefault <- Default
glm.fit <- glm(default ~ student, data = dfDefault, family = binomial)
summary(glm.fit)
```

```
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.50413    0.07071  -49.55  < 2e-16 ***
studentYes   0.40489    0.11502    3.52 0.000431 ***
```

$$P(default = yes\ | student = \boldsymbol{yes}) = \frac{e^{-3.5041+0.4049*\mathbf{1}}}{1+e^{-3.5041+0.4049*\mathbf{1}}} = .0431$$

$$P(default = yes\ | student = \boldsymbol{no}) = \frac{e^{-3.5041+0.4049*\mathbf{0}}}{1+e^{-3.5041+0.4049*\mathbf{0}}} = .0292$$




```
> (exp(-3.5041+ (0.4049 *1)))/ (1 + exp(-3.5041+ (0.4049 *1)))
[1] 0.04314027
> (exp(-3.5041+ (0.4049 * 0)))/ (1 + exp(-3.5041+ (0.4049 * 0)))
[1] 0.0291958
```

Hints:

when you get into equations and many modeling algorithms, you'll find that
variable values need to be integers (0 and 1 only), but not always (some algorithms want factors or text). How do you know? RTFM

BTW, factors will often convert to 1, 2 if you use as.integer, so:

```
unique(as.numeric(dfDefault[,"default"])) # check to make sure you only have 2 values (1, 2), then

dfDefault$default <- as.integer(dfDefault$default)-1
```

If the data is given in factors or text, then you will have to convert. Even if it looks like “0” and “1”, it won’t covert to 0 and 1. Something like Emp_Turn$Left <- as.integer(Emp_Turn$Left)-1 should work [Homework]

This is how I usually pull samples:

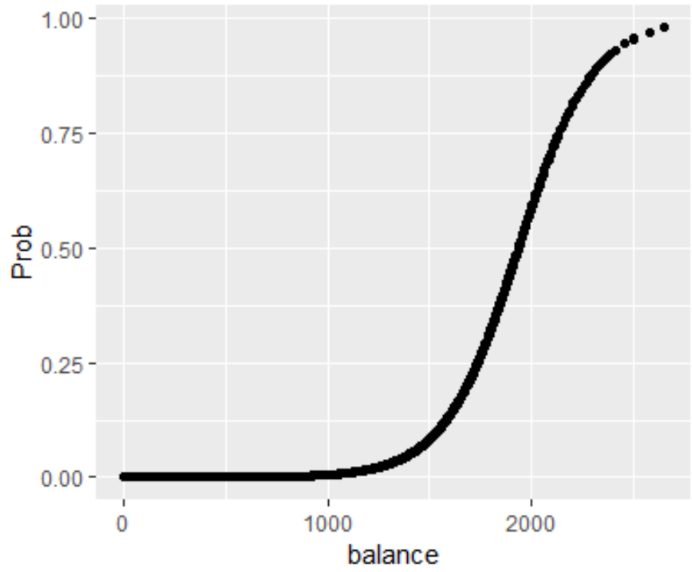
```
dfDefault <- dfDefault %>% rownames_to_column("SampleID")
train <- sample_n(dfDefault, round(nrow(dfDefault)*.6,0))
test <- dfDefault %>% anti_join(train, by = "SampleID")
```

Algorithms do REALLY well if they’ve seen the data before – duh! If you need a realistic estimate, you hold out a validation (test) set and be careful not use any of that data for training. Be careful – you can embarrass yourself here!

Sampling is a BIG DEAL and we’ll spend a lot of time on it.

Modeling one continuous variable

```
> glm.fit <- glm(default ~ balance, data = dfDefault, family = binomial)
> summary(glm.fit)

Call:
glm(formula = default ~ balance, family = binomial, data = dfDefault)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2697  -0.1465  -0.0589  -0.0221   3.7589

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.065e+01  3.612e-01  -29.49   <2e-16 ***
balance      5.499e-03  2.204e-04   24.95   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2920.6  on 9999  degrees of freedom
Residual deviance: 1596.5  on 9998  degrees of freedom
AIC: 1600.5

Number of Fisher Scoring iterations: 8

>
> dfDefault$Prob <- predict(glm.fit, type = "response")
> ggplot(dfDefault, aes(x=balance, y=Prob)) + geom_point()
> glm.fit <- glm(default ~ student, data = dfDefault, family = binomial)
```



Multiple Logistic Regression

```
glm(formula = default ~ student + balance + income, family = binomial,
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2314  -0.1351  -0.0509  -0.0174   3.5987

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.153e+01  6.797e-01 -16.958   <2e-16 ***
studentYes  -5.461e-01  3.148e-01  -1.735   0.0827 .
balance      6.039e-03  3.195e-04  18.899   <2e-16 ***
income       6.608e-06  1.089e-05   0.607   0.5440
---
```

Categorical variables handled the same way we did with regression

```
test$mProb2 <- predict(glMod2, type = "response", newdata = test)
mTest = model.matrix(default ~ student + balance + income, data = test)
bet1 <- as.numeric(glMod2$coefficients)
test$tmProb2 <- exp( t(bet1%*%t(mTest)))/(1+exp(t(bet1%*%t(mTest))))
df2$mProb <- predict(mglm.fit, type = "response")
```

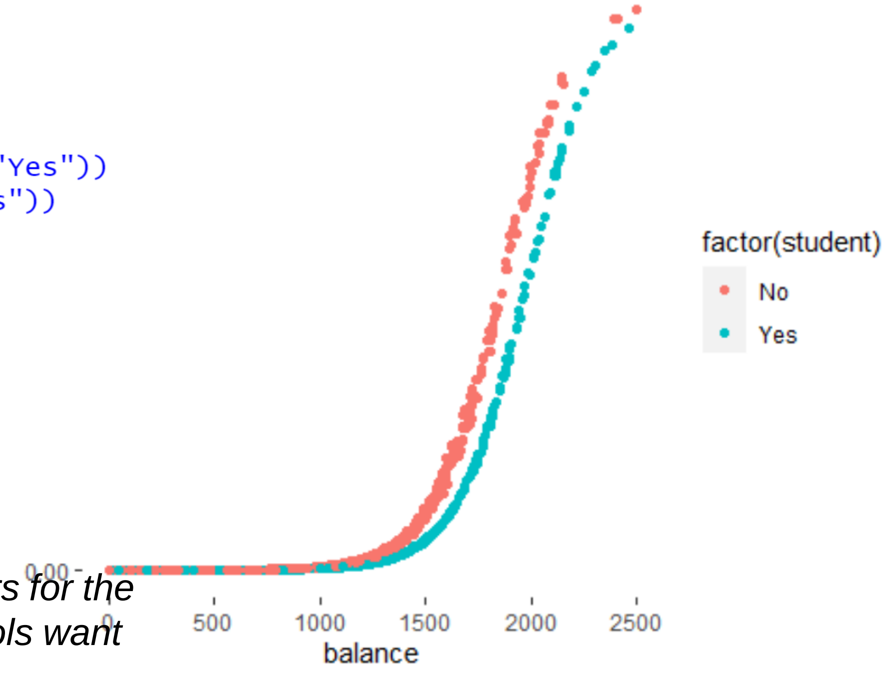
Notice that I'm using a test matrix, but writing predictions to test dataset

$$P(y) = \frac{e^{\beta_0+\beta_1 X}}{1 + \ e^{\beta_0+\beta_1 X}}$$

Like lm, we can get the coefficients from glm, and use the predict function (a little different – note the parameters) And we can create a model matrix from the data and predict using a model matrix and linear algebra

```r
ggplot(test, aes(x=balance, y=tmProb2, color = factor(student))) +
  geom_point() +
  theme(panel.background = element_rect(fill = "white"))
df2$mProb <- predict(mglm.fit, type = "response")
```

```r
# how did we do?

test$class = factor(if_else(test$tmProb2 < .5, "No", "Yes"))
test$D2 = factor(if_else(test$default < .5, "No", "Yes"))

table(test$class, test$D2)

      No  Yes
No  3844   87
Yes   23   46
```



So, while many of the modeling algorithms want integers for the response variable, many of the subsequent analysis tools want factors. How do you know? RTFM

```
> confusionMatrix((test$class), factor(test$D2), positive = "Yes")
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  3844   87
       Yes   23   46

               Accuracy : 0.9725
                 95% CI : (0.9669, 0.9773)
    No Information Rate : 0.9668
    P-Value [Acc > NIR] : 0.02126

                  Kappa : 0.4428

 Mcnemar's Test P-Value : 1.892e-09

            Sensitivity : 0.34586
            Specificity : 0.99405
         Pos Pred Value : 0.66667
         Neg Pred Value : 0.97787
             Prevalence : 0.03325
         Detection Rate : 0.01150
   Detection Prevalence : 0.01725
      Balanced Accuracy : 0.66996

       'Positive' Class : Yes
```

This looks good at first, but look closer.

The True Positives, ***Sensitivity***, is at is at 34%, so 65% of applicants that are predicted to default, would not. This is a false positive and it costs you business *(because you would reject applicants that would be good customers).*

On the other side, we have 99% ***Specificity,*** True Negatives. False negatives here would result in bad debt expense. Note: bad debt expense is a balance in business – too little is just as bad as too much.

There are ways to tune sampling and improve responses which we'll study soon.

BAUER COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

Confusion Matrix

```
> confusionMatrix((test$class), factor(test$D2), positive = "Yes")
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  3844   87
       Yes   23   46

               Accuracy : 0.9725
                 95% CI : (0.9669, 0.9773)
    No Information Rate : 0.9668
    P-Value [Acc > NIR] : 0.02126

                  Kappa : 0.4428

 Mcnemar's Test P-Value : 1.892e-09

            Sensitivity : 0.34586
            Specificity : 0.99405
         Pos Pred Value : 0.66667
         Neg Pred Value : 0.97787
             Prevalence : 0.03325
         Detection Rate : 0.01150
   Detection Prevalence : 0.01725
      Balanced Accuracy : 0.66996

       'Positive' Class : Yes
```

		Actual	
		Negative	Positive
Predicted	Negative	True Negative	False Negative
	Positive	False Positive	True Positive

Sensitivity (also called the **true positive rate**) measures the proportion of positives that are correctly identified. 46/(46+87) = **.34..**

Specificity (also called the **true negative rate**) measures the proportion of negatives that are correctly identified. 3844/(3844+23) = **.99…**

Prevalence = *(87+46)/(87+46+3844+23) = .033... Total Pos in Sample*

Positive Pred Value = *(sensitivity * prevalence)/((sensitivity*prevalence) + ((1-specificity)*(1-prevalence))) =*
*=(0.34586*0.03325)/((0.34586*0.03325)+((1-0.99405)*(1-0.03325)))*
= .666 *(est % of predicted positives that were correctly identified 46/(46+23) for rough)*

Neg Pred Value = *(specificity * (1-prevalence))/(((1-sensitivity)*prevalence) + ((specificity)*(1-prevalence)))... etc.*

Multinomial Logistic Regression

```
setwd("C:/Users/ellen/Documents/Spring 2019/DA2/Section 1/Classification/Data")
prog <- read.csv("programs.csv")
prog$prog2 <- relevel(prog$prog, ref = "academic")
fit.prog <- vglm(prog ~ math, family = multinomial, data = prog)
coef(fit.prog, matrix = TRUE)
```

```
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept):1 -7.19172    1.33778  -5.376 7.62e-08 ***
(Intercept):2 -3.13613    1.36231  -2.302   0.0213 *
math:1         0.15497    0.02676   5.792 6.95e-09 ***
math:2         0.06296    0.02800   2.249   0.0245 *
---
```

A multinomial logit model generalizes LogReg to a multiclass model. In simple models, we create a **reference *(or pivot)*** outcome, and all the rest of the nominal probabilities are independently regressed against that reference.

```
> vglmP <- predictvglm(fit.prog, type = "response")
> tstRec <- prog[1,]
> L1 <- fit.prog@coefficients[1] + fit.prog@coefficients[3]*tstRec[8]
> L2 <- fit.prog@coefficients[2] + fit.prog@coefficients[4]*tstRec[8]
> denom <- 1 + exp(L1) + exp(L2)
> pihat1 <- exp(L1)/denom
> pihat2 <- exp(L2)/denom
> pihat3 <- 1/denom
>
> tst <- rbind(vglmP[1,], c(pihat1, pihat2, pihat3))
> tst
     academic   general  vocation
[1,] 0.2155953 0.2861312 0.4982735
[2,] 0.2155953 0.2861312 0.4982735
```

$$P_1 = \frac{e^{L1}}{1+e^{L1}+e^{L2}}$$

$$P_2 = \frac{e^{L2}}{1+e^{L1}+e^{L2}}$$

$$P_3 = \frac{1}{1+e^{L1}+e^{L2}}$$

$$P(program = academic\,|math = 41) = \frac{e^{-7.19172+0.15497*\mathbf{41}}}{1+e^{-7.19172+0.15497*\mathbf{41}}+e^{-3.13613+0.06296*\mathbf{41}}} = 0.2155953$$

$$P(program = general\,|math = 41) = \frac{e^{-3.13613+0.6296*\mathbf{41}}}{1+e^{-7.19172+0.15497*\mathbf{41}}+e^{-3.13613+0.06296*\mathbf{41}}} = 0.2861312$$

$$P(program = vocation\,|math = 41) = \frac{1}{1+e^{-7.19172+0.15497*\mathbf{41}}+e^{-3.13613+0.06296*\mathbf{41}}} = 0.4982735$$

Lets review what we're saying here. Given a math score of 41 *(the lowest score)*

```
> unique(prog$math)
 [1] 41 44 42 40 46 33 38 37 39 43 45 49 47 57 50 52 48 54 53 51 55 61 56 35 59 66 58 60 63 64 62
[32] 67 65 72 69 70 68 75 71 73
```

What's the probability the student is in an academic, general, or vocation program?

$$P(program = academic \mid math = 41) = \frac{e^{-7.19172+0.15497 * \mathbf{41}}}{1+ e^{-7.19172+ 0.15497 * \mathbf{41}} + e^{-3.13613+0.06296 * \mathbf{41}}} = 0.2155953$$

$$P(program = general \quad \mid math = 41) = \frac{e^{-3.13613+0.6296 * \mathbf{41}}}{1+ e^{-7.19172+ 0.15497 * \mathbf{41}} + e^{-3.13613+0.06296 * \mathbf{41}}} = 0.2861312$$

$$P(program = vocation \quad \mid math = 41) = \frac{1}{1+ e^{-7.19172+ 0.15497 * \mathbf{41}} + e^{-3.13613+0.06296 * \mathbf{41}}} = 0.4982735$$

Expanding this model to multiple predictors, the model produces probabilities for each line, L, for each nominal outcome

```
> fit.prog <- vglm(prog ~ ses + write, family = multinomial, data = prog)
> vglmP <- predictvglm(fit.prog, type = "response")
> prog$Predict <-  colnames(vglmP)[max.col(vglmP,ties.method="first")]
> table(prog$Predict, prog$prog2)

          academic general vocation
 academic       92      27       23
 general         4       7        4
 vocation        9      11       23
```

	academic	general	vocation
1	0.1482781	0.3382488	0.51347306
2	0.1202034	0.1806286	0.69916808
3	0.4186789	0.2368082	0.34451282
4	0.1726902	0.3508414	0.47646847
5	0.1001247	0.1689379	0.73093743
6	0.3533612	0.2377981	0.40884067

prog2	Predict
vocation	vocation
general	vocation
vocation	academic
vocation	vocation
vocation	vocation
general	vocation
vocation	vocation
vocation	vocation
vocation	vocation
vocation	vocation

We're using vglm from the VGAM package here because it has a multinomial version of glm. This is not the most flexible approach to multinomial *(or multiclass)* analysis, and non-parametric algorithms will usually produce a lower error *(which doesn't mean it's better – remember the interpretability/flexibility tradeoff)*. It's almost always good baseline and extends conceptually into Bayesian multinomial modeling.

Just reviewing: we studied a GAM last week, which is a type of GLM that uses different functions within knots to fit data. It also uses a link function, which is the basis of the GLM:

```
> prog %>% group_by(prog) %>% summarise(Cnt = n())
# A tibble: 3 x 2
  prog        Cnt
  <chr>     <int>
1 academic    105
2 general      45
3 vocation     50
```

actual

predicted

	academic	general	vocation	
academic	92	27	23	142
general	4	7	4	15
vocation	9	11	23	43
	105	45	50	200
	87.6%			a
		15.6%		
			46.0%	

This is just a speadsheet. You can't use a confusion matrix with mulitnomials

There are several algorithms that can adapt to multinomial *(usually called multi-class)* problems. These are common in accounting and transaction analysis *(where you're testing account distribution).*

That said, these are hard problems to solve and predicting > 50 classes required clever solution design

Logistic Regression Exercise

Using the quote history data, build a logistic regression model to predict whether an opportunity will result in a Win or Loss based on data about price, competition, ATP and customer requirements

```
quoteData <- filter(quoteData, Result %in% c(0, 1))
quoteData <- quoteData %>% rownames_to_column("SampleID")
quoteData$SampleID  <- as.numeric(quoteData$SampleID)
quoteData$QuoteDiff <- quoteData$QuoteDiff/1000
quoteData$RSF <- factor(quoteData$RSF)
train <- sample_n(quoteData, nrow(quoteData)-100)
test <- quoteData %>% anti_join(train, by = "SampleID")
df2$mProb <- predict(mglm.fit, type = "response")
```

Convert and scale quote vs competitor quote to scale with difference in quotes (good practice always)

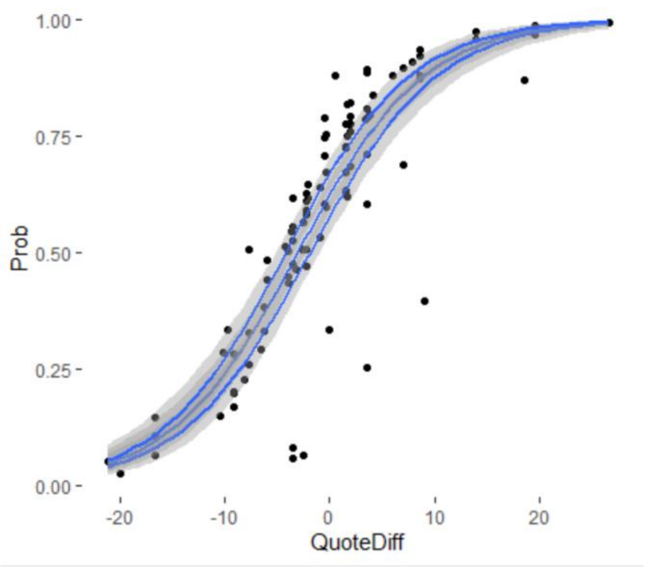
Notice that we convert absolute values to difference (e.g., quote difference, date difference, etc.) Algorithms are more sensitive to differences than absolute values. The data dimensions are:

QuoteDiff – difference between competitor and company quote
RFPDiff – difference in days between date RFP return requested and actual response date
ATPDiff – difference between date equipment required, and data Available to Promise.
RSF is an index of the Relationship Strength Factor

```
glm(formula = Result ~ RSF + QuoteDiff + RFPDiff + ATPDiff, family = binomial,
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7244  -0.8235   0.3689   0.8338   2.5483

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.163756   0.436883  -4.953 7.32e-07 ***
RSF2         2.778560   0.556490   4.993 5.94e-07 ***
RSF3         2.427742   0.475009   5.111 3.21e-07 ***
RSF4         2.893989   0.460893   6.279 3.41e-10 ***
QuoteDiff    0.186845   0.017025  10.975  < 2e-16 ***
RFPDiff      0.043908   0.014934   2.940  0.00328 **
ATPDiff      0.016091   0.003973   4.050 5.13e-05 ***
```

A little different with glm. The Prediction is an object you create, then pull variables and metrics out. Here, we pull the probability and se out.

```
testPred <- predict(glm.fit, type = "response", newdata = test, se.fit = T)

test$Prob <- testPred$fit
test$lcl <- test$Prob - testPred$se.fit
test$ucl <- test$Prob + testPred$se.fit
```

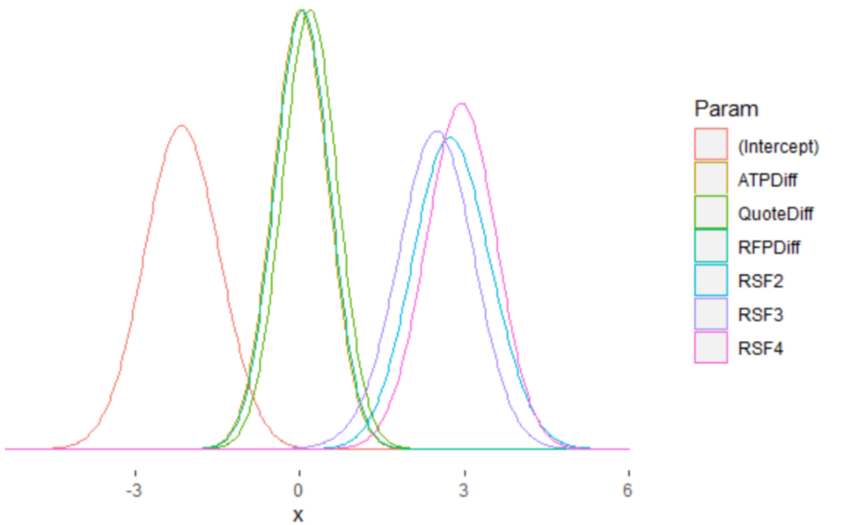


BAUER
COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

It's important to keep in mind that parameters are estimates, and the value we often refer to is the mean. And the confidence *(or credibility)* of our estimate varies widely. You can see visually that we have more confidence in the QuoteDiff parameters than we do RSF parameters.

```
> confint.default(glm.fit) # this uses likelihood to compute Wald CIs
nal symmetric)
                  2.5 %      97.5 %
(Intercept) -3.020030110 -1.30748204
RSF2         1.687860007  3.86925966
RSF3         1.496741811  3.35874188
RSF4         1.990654118  3.79732324
QuoteDiff    0.153476315  0.22021422
RFPDiff      0.014637866  0.07317852
ATPDiff      0.008303001  0.02387866
```

```
GLMParamEst <- data.frame(mean = glm.fit$coefficients, sdEst =
  (confint.default(glm.fit)[,2]-glm.fit$coefficients)/1.96)
GLMParamEst <- rownames_to_column(GLMParamEst, "Param")
PlotData <- data.frame(Param = GLMParamEst$Param,
  x = rnorm(700, GLMParamEst$mean, GLMParamEst$sdEst))

ggplot(PlotData, aes(x = x, color = Param)) +
  geom_density(bw = .5) +
  scale_x_continuous(limits = c(-6, 6)) +
  theme(panel.background = element_rect(fill = "white"))
```



Backing into the CIs – algorithm uses 95%, which is 1.96 sd

This is what we're after – the parameters and the confidence intervals. Once we have these, we can plug them into an array of analyses and applications. As I've said before, most applications do NOT use packaged predict functions – the world is too complex, and transaction / data scale and dynamics are too high.

BAUER COLLEGE OF BUSINESS
UNIVERSITY of HOUSTON

```
tst1 <- model.matrix(Result ~ RSF + QuoteDiff + RFPDiff + ATPDiff,  data = test)
bet1 <- as.numeric(glm.fit$coefficients)
test$tmProb2 <- exp( t(bet1%*%t(tst1)))/(1+exp(t(bet1%*%t(tst1))))

# show that equation gets same result as glm
sum(round(test$Prob - test$tmProb2,0))
1] 0
# score results
test$PResult <- ifelse(test$Prob < .5, 0, 1)
# check metrics
confusionMatrix(factor(test$PResult) , factor(test$Result))
```

$$P(y) = \frac{e^{\beta_0+\beta_1 X}}{1+\ e^{\beta_0+\beta_1 X}}$$

```
          Reference
Prediction  0  1
         0 28  5
         1 14 53

               Accuracy : 0.81
                 95% CI : (0.7193, 0.8816)
    No Information Rate : 0.58
    P-Value [Acc > NIR] : 9.183e-07

                  Kappa : 0.5981

 Mcnemar's Test P-Value : 0.06646

            Sensitivity : 0.6667
            Specificity : 0.9138
         Pos Pred Value : 0.8485
         Neg Pred Value : 0.7910
             Prevalence : 0.4200
         Detection Rate : 0.2800
   Detection Prevalence : 0.3300
      Balanced Accuracy : 0.7902

       'Positive' Class : 0
```

Here, we're converting probabilities *(the outcome of the equation)* to categories *(0, 1)*.

This is an important point – we can decide the level of probability breaks *(.5 is common in binomial models, but it doesn't have to be that way – as we'll see later)*

Again, *there are many things we can do with tuning and resampling, which we'll study in the next couple of sections*