SCORECARD CREATION

Dr. Aric LaBarr
Institute for Advanced Analytics

Process Flow

Data Collection
- Variable Selection
- Sample Size
- Sample / Performance Window

Data Cleaning
- Eliminate Duplicates
- Examine / Remove Outliers

Variable Grouping and Selection
- Weights of Evidence (WOE)
- Information Value (IV)
- Gini Criterion

Initial Scorecard Creation
- Logistic Regression
- Accuracy
- Threshold
- Assessment

Reject Inference
- Remove bias resulting from exclusion of rejects

Final Scorecard Creation
- Final Model Assessment

INITIAL SCORECARD CREATION

Initial Scorecard Model

- The scorecard is (typically) based on a logistic regression model:

$$logit(p) = \log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$

- *p* is the posterior probability of default (PD) given the inputs.

Blasphemy!!!

- Wait, so I’m going through all that math just to throw things back into the logistic regression I was trying to avoid in the first place?!?!?!?!

Different Inputs

- Instead of using the original variables for the model, scorecard models have the binned variables as their foundation.

Observation	Target	Bureau Score	Bureau Score Bin (R)	Bureau Score WOE (R)
1	0	757	716 – 765	1.0914
2	1	NA	Missing	-0.6972
3	0	626	605 – 629	-0.9586
4	0	693	665 – 716	0.1776
5	0	706	665 – 716	0.1776
6	0	673	665 – 716	0.1776
7	0	730	716 – 765	1.0914

Different Inputs

- Instead of using the original variables for the model, scorecard models have the binned variables as their foundation.

Actual input into model!

Observation	Target	Bureau Score	Bureau Score Bin (R)	Bureau Score WOE (R)
1	0	757	716 – 765	1.0914
2	1	NA	Missing	-0.6972
3	0	626	605 – 629	-0.9586
4	0	693	665 – 716	0.1776
5	0	706	665 – 716	0.1776
6	0	673	665 – 716	0.1776
7	0	730	716 – 765	1.0914

Different Inputs

- Instead of using the original variables for the model, scorecard models have the binned variables as their foundation.

Actual input into model!

- Inputs are still treated as continuous.
- All variables now on same scale.
- Model **coefficients** are desired output for the scorecard.
- **Coefficients** now serve as measures of variable importance.

Bureau Score WOE (R)
1.0914
-0.6972
-0.9586
0.1776
0.1776
0.1776
1.0914

Different Input – Example

```
smbinning.gen(df = train, ivout = result_all_sig$bureau_score, chrname = "bureau_score_bin")
```

Different Input – Example

```
smbinning.gen(df = train, ivout = result_all_sig$bureau_score, chrname = "bureau_score_bin")
```

Observation	Target	Bureau Score	Bureau Score Bin (R)	Bureau Score WOE (R)
1	0	757	716 – 765	1.0914
2	1	NA	Missing	-0.6972
3	0	626	605 – 629	-0.9586
4	0	693	665 – 716	0.1776
5	0	706	665 – 716	0.1776
6	0	673	665 – 716	0.1776
7	0	730	716 – 765	1.0914

Different Input – Example

```
for (i in 1:length(train)) {
  bin_name <- "bureau_score_bin"
  bin <- substr(train[[bin_name]][i], 2, 2)

  woe_name <- "bureau_score_WOE"

  if(bin == 0) {
    bin <- dim(result_all_sig$bureau_score$ivtable)[1] - 1
    train[[woe_name]][i] <- result_all_sig$bureau_score$ivtable[bin, "WoE"]
  } else {
    train[[woe_name]][i] <- result_all_sig$bureau_score$ivtable[bin, "WoE"]
  }
}
```

Different Input – Example

Observation	Target	Bureau Score	Bureau Score Bin (R)	Bureau Score WOE (R)
1	0	757	716 – 765	1.0914
2	1	NA	Missing	-0.6972
3	0	626	605 – 629	-0.9586
4	0	693	665 – 716	0.1776
5	0	706	665 – 716	0.1776
6	0	673	665 – 716	0.1776
7	0	730	716 – 765	1.0914

Initial Scorecard – Example

```
# Build Initial Scorecard #
initial_score <- glm(data = train, bad ~ tot_derog_WOE +
                                         tot_tr_WOE +
                                         age_oldest_tr_WOE +
                                         tot_rev_line_WOE +
                                         rev_util_WOE +
                                         bureau_score_WOE +
                                         down_pyt_WOE +
                                         ltv_WOE,
                     weights = train$weight, family = "binomial")

summary(initial_score)
```

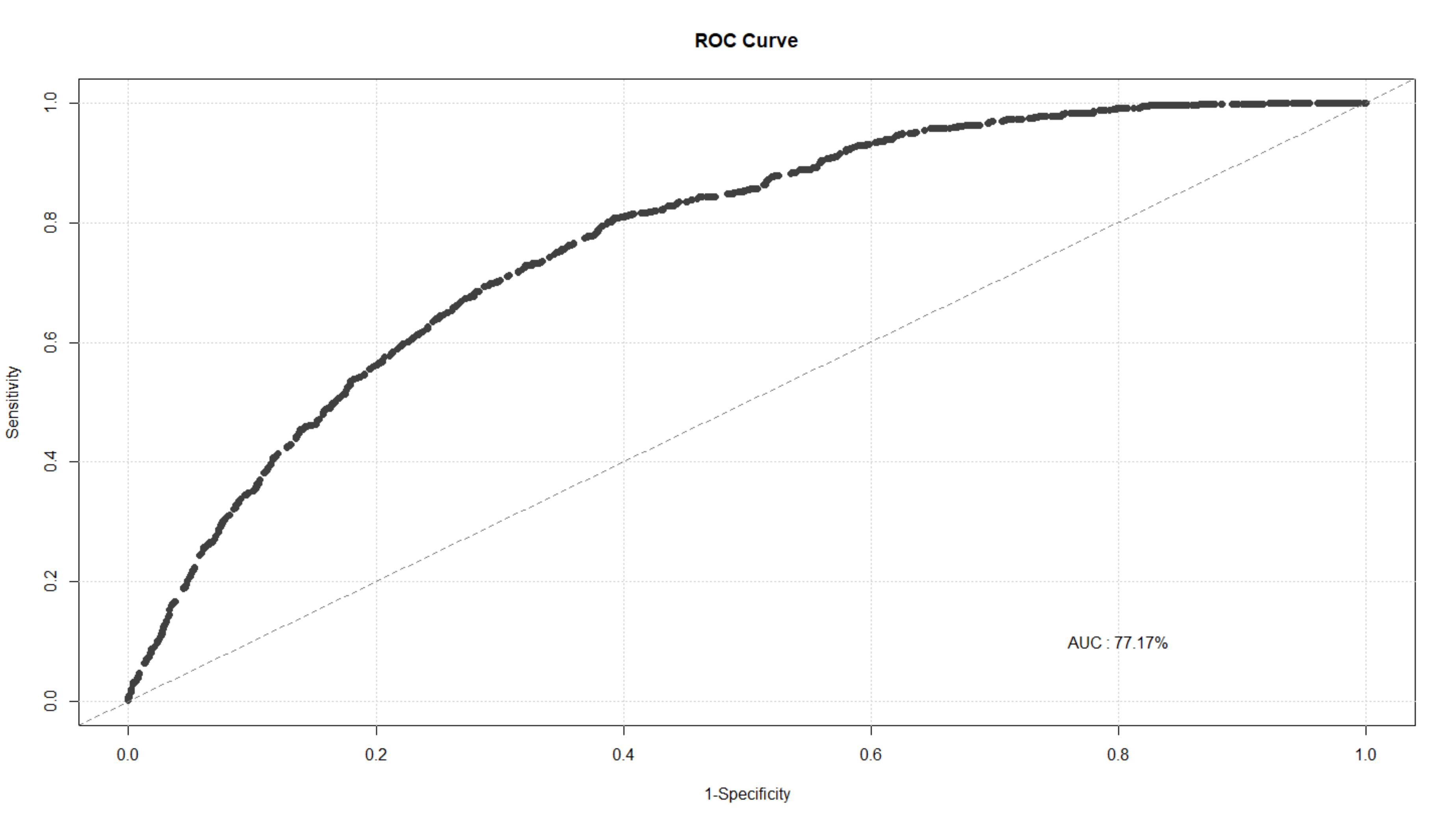
Initial Scorecard – Example

```
> summary(initial_score)

Call:
glm(formula = bad ~ age_oldest_tr_WOE + tot_rev_line_WOE + rev_util_WOE +
    bureau_score_WOE + down_pyt_WOE + ltv_WOE, family = "binomial",
    data = train, weights = train$weight)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6599  -0.7457  -0.4245  -0.1655   3.3717

Coefficients:
                  Estimate Std. Error z value             Pr(>|z|)
(Intercept)       -2.98073    0.04098 -72.729 < 0.0000000000000002 ***
age_oldest_tr_WOE -0.25059    0.08328  -3.009              0.00262 **
tot_rev_line_WOE  -0.43244    0.07139  -6.057        0.00000000138 ***
rev_util_WOE      -0.23988    0.07647  -3.137              0.00171 **
bureau_score_WOE  -0.81323    0.05398 -15.064 < 0.0000000000000002 ***
down_pyt_WOE      -0.38994    0.14825  -2.630              0.00853 **
ltv_WOE           -0.86347    0.10111  -8.540 < 0.0000000000000002 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6910.4  on 4376  degrees of freedom
Residual deviance: 6083.5  on 4370  degrees of freedom
AIC: 6184.2

Number of Fisher Scoring iterations: 6
```

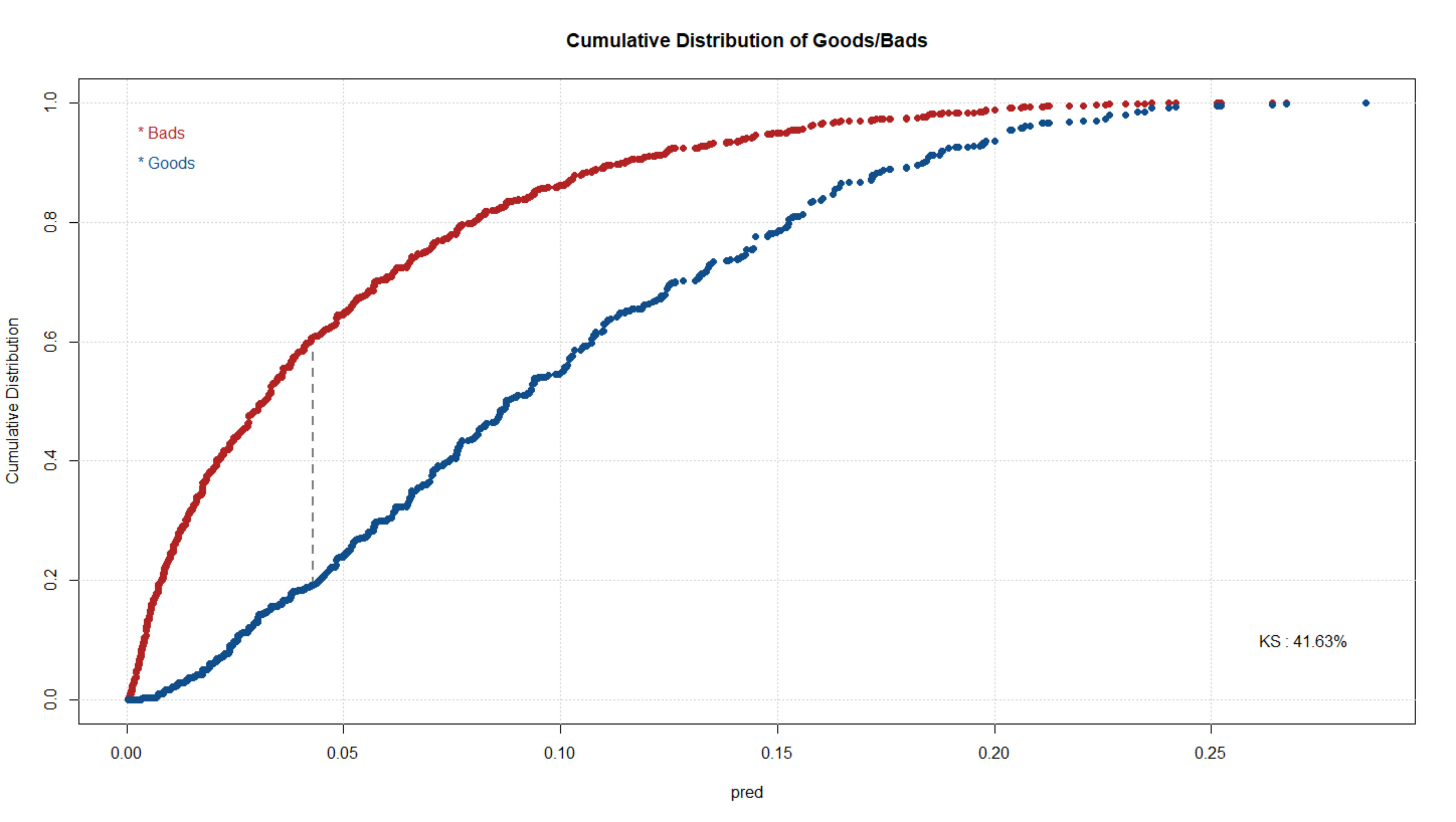
Model Evaluation

- Variable significance – review using “standard” output of logistic regression, but don’t forget **business logic**.
- Overall performance of model – AUC (area under ROC curve, also called c) is the most popular criterion.
- This is only a **preliminary scorecard**.
- Final scorecard is created after reject inference is performed.

Model Evaluation – Example



Model Evaluation – Example



?

SCALING THE SCORECARD

Scaling the Scorecard

- The relationship between odds and scores is represented by a linear function:

$$Score = Offset + Factor \times \log(odds)$$

- If the scorecard is developed using “odds at a certain score” and “points to double the odds” (PDO), $Factor$ and $Offset$ can be calculated using the simultaneous equations:

$$Score = Offset + Factor \times \log(odds)$$

$$Score + PDO = Offset + Factor \times \log(2 \times odds)$$

Scaling the Scorecard

- Solving the equations for PDO, you get the following results:

$$PDO = Factor \times \log(2)$$

- Therefore,

$$Factor = \frac{PDO}{\log(2)}$$

$$Offset = Score - Factor \times \log(odds)$$

Scaling the Scorecard – Example

- If a scorecard were scaled where the developer wanted odds of 50:1 at 600 points and wanted the odds to double every 20 points (PDO = 20), *Factor* and *Offset* would be:

$$Factor = \frac{20}{\log(2)} = 28.8539$$

$$Offset = 600 - (28.8539 \times \log(50)) = 487.123$$

- Therefore, each score corresponding to each set of odds can be calculated as follows:

$$Score = 487.123 + 28.8539 \times \log(odds)$$

Scaling the Scorecard – Example

- If a scorecard were scaled where the developer wanted odds of 50:1 at 600 points and wanted the odds to double every 20 points (PDO = 20), *Factor* and *Offset* would be:

$$Factor = \frac{20}{\log(2)} = 28.8539$$

$$Offset = 600 - (28.8539 \times \log(50)) = 487.123$$

- Therefore, each score corresponding to each set of odds can be calculated as follows:

$$Score = 487.123 + 28.8539 \times \boxed{\log(odds)}$$

Why people still love logistic regression!

Scaling the Scorecard – Example

- If a scorecard were scaled where the developer wanted odds of 50:1 at 600 points and wanted the odds to double every 20 points (PDO = 20), *Factor* and *Offset* would be:

$$Factor = \frac{20}{\log(2)} = 28.8539$$

$$Offset = 600 - (28.8539 \times \log(50)) = 487.123$$

- Therefore, each score corresponding to each set of odds can be calculated as follows:

$$Score = 487.123 + 28.8539 \times \boxed{\log(odds)}$$

This is predicted value from logit function.

Scaling the Scorecard – Example

Score	Odds
600	50.0
601	51.8
604	57.4
.	
.	
.	
.	
620	100.0

Points Allocation

- The points allocated to attribute *i* of characteristic *j* are computed as follows:

$$Points_{i,j} = -\left(WOE_{i,j} \times \hat{\beta}_j + \frac{\hat{\beta}_0}{L}\right) \times Factor + \frac{Offset}{L}$$

- $WOE_{i,j}$: Weight of evidence for attribute *i* of characteristic *j*
- $\hat{\beta}_j$: Regression coefficient for characteristic *j*
- $\hat{\beta}_0$: Intercept term from model
- L: Total number of characteristics
- Points typically rounded to nearest integer.

Points Allocation – Example

```
# Add Scores to Initial Model #
pdo <- 20
score <- 600
odds <- 50
fact <- pdo/log(2)
os <- score - fact*log(odds)
var_names <- names(initial_score$coefficients[-1])

for(i in var_names) {
  beta <- initial_score$coefficients[i]
  beta0 <- initial_score$coefficients["(Intercept)"]
  nvar <- length(var_names)
  WOE_var <- train[[i]]
  points_name <- paste(str_sub(i, end = -4), "points", sep="")

  train[[points_name]] <- -(WOE_var*(beta) + (beta0/nvar))*fact + os/nvar
}

colini <- (ncol(train)-nvar + 1)
colend <- ncol(train)
train$Score <- rowSums(train[, colini:colend])

hist(train$Score, breaks = 50, main = "Distribution of Scores", xlab = "Score")
```

REJECT INFERENCE

Process Flow

Data Collection
- Variable Selection
- Sample Size
- Sample / Performance Window

Data Cleaning
- Eliminate Duplicates
- Examine / Remove Outliers

Variable Grouping and Selection
- Weights of Evidence (WOE)
- Information Value (IV)
- Gini Criterion

Initial Scorecard Creation
- Logistic Regression
- Accuracy
- Threshold
- Assessment

Reject Inference
- Remove bias resulting from exclusion of rejects

Final Scorecard Creation
- Final Model Assessment

Reject Inference

- **Reject inference** is the process of inferring the status of the rejected applicants based on the accepted applicants model in an attempt to use their information to build a scorecard that is representative of the entire applicant population.
- Reject inference is about solving sample bias so that the development sample is similar to the population to which the scorecard will be applied.

Rejected Inference

- Can we develop a scorecard without rejected applications? **YES!**
- Is it **legally permissible** to develop a scorecard without rejected applications? **YES!**
- If yes, then how **biased** would the scorecard model be? **DEPENDS!**

- *"My suggestion is to develop the scorecard using what data you have, but start saving rejected applications ASAP."*
  Raymond Anderson, Head of Scoring at Standard Bank Africa, South Africa

Why Reject Inference?

- Initial scorecard used only **known** good and bad loans (accepted applicants only) – also called “behavioral scoring”
- Reduce bias in model and provide risk estimates for the “through-the-door” population – also called “application scoring”
- Comply with regulatory requirements (FDIC, Basel)
- Provide a scorecard that is able to generalize better to the entire credit application population.

Reject Inference Techniques

- Three common techniques for reject inference:
  1. Hard Cutoff Augmentation
  2. Parceling Augmentation
  3. Fuzzy Augmentation (DEFAULT in SAS)

Hard Cutoff Augmentation

1. Build a scorecard model using the known good/bad population (accepted applications)
2. Score the rejected applications with this model to obtain each rejected applicant's probability of default and their score on the scorecard model.
3. Create weighted cases for the rejected applicants – weight applied is the "rejection rate" which adjusts the number of sampled rejects to accurately reflect the number of rejects from population.

Hard Cutoff Augmentation

4. Set a cut-off score level above which applicant is deemed good and below applicants deemed bad.
5. Add inferred goods and bads with known goods and bads and rebuild scorecard.

Hard Cutoff Augmentation – Example

```
# Reject Inference - Hard Cut-off #
rejects_scored$pred <- predict(initial_score, newdata=rejects_scored, type='response')

rejects$bad <- as.numeric(rejects_scored$pred > 0.0603)
rejects$weight <- ifelse(rejects$bad == 1, 2.80, 0.59)
rejects$good <- abs(rejects$bad - 1)

comb_hard <- rbind(accepts, rejects) # New Combined Data Set #
```

Hard Cutoff Augmentation – Example

```
# Reject Inference - Hard Cut-off #
rejects_scored$pred <- predict(initial_score, newdata=rejects_scored, type='response')

rejects$bad <- as.numeric(rejects_scored$pred > 0.0603)
rejects$weight <- ifelse(rejects$bad == 1, 2.80, 0.59)
rejects$good <- abs(rejects$bad - 1)

comb_hard <- rbind(accepts, rejects) # New Combined Data Set #
```

Hard Cutoff Augmentation – Example

```
# Reject Inference - Hard Cut-off #
rejects_scored$pred <- predict(initial_score, newdata=rejects_scored, type='response')

rejects$bad <- as.numeric(rejects_scored$pred > 0.0603)
rejects$weight <- ifelse(rejects$bad == 1, 2.80, 0.59)
rejects$good <- abs(rejects$bad - 1)

comb_hard <- rbind(accepts, rejects) # New Combined Data Set #
```

Parceling Augmentation

1. Build a scorecard model using the known good/bad population (accepted applications)
2. Score the rejected applications with this model to obtain each rejected applicant's probability of default and their score on the scorecard model.
3. Create weighted cases for the rejected applicants – weight applied is the "rejection rate" which adjusts the number of sampled rejects to accurately reflect the number of rejects from population.

Parceling Augmentation

4. Define score ranges manually or automatically with simple bucketing.
5. The inferred good/bad status of the rejected applicants will be assigned **randomly** and proportional to the number of goods and bads in the accepted population within each score range.
6. If desired, apply the event rate increase factor to P(bad) to increase the proportion of bads among the rejects (oversampling with the rejects)
7. Add the inferred goods and bads back in with the known goods and bads and rebuild the scorecard.

Parceling Augmentation – Example

	Accepted Applicants				Rejected Applicants		
Score Range	# Bad	# Good	% Bad	%Good	Rejects	# Inferred Bad	# Inferred Good
< 655	0	0	0%	0%	5	?	?

Parceling Augmentation – Example

	Accepted Applicants				Rejected Applicants		
Score Range	# Bad	# Good	% Bad	%Good	Rejects	# Inferred Bad	# Inferred Good
< 655	0	0	0%	0%	5	5	0

Assume bad if no information to prove otherwise

Parceling Augmentation – Example

	Accepted Applicants				Rejected Applicants		
Score Range	# Bad	# Good	% Bad	%Good	Rejects	# Inferred Bad	# Inferred Good
< 655	0	0	0%	0%	5	5	0
655 – 665	300	360	45.5%	54.5%	190	?	?

Parceling Augmentation – Example

	Accepted Applicants				Rejected Applicants		
Score Range	# Bad	# Good	% Bad	%Good	Rejects	# Inferred Bad	# Inferred Good
< 655	0	0	0%	0%	5	5	0
655 – 665	300	360	45.5%	54.5%	190	86	?

$$\boxed{\mathbf{0.455 \times 190 \approx 86}}$$

Randomly assign!

Parceling Augmentation – Example

	Accepted Applicants				Rejected Applicants		
Score Range	# Bad	# Good	% Bad	%Good	Rejects	# Inferred Bad	# Inferred Good
< 655	0	0	0%	0%	5	5	0
655 – 665	300	360	45.5%	54.5%	190	86	114

$$190 - 86 = 114$$

Parceling Augmentation – Example

	Accepted Applicants				Rejected Applicants		
Score Range	# Bad	# Good	% Bad	%Good	Rejects	# Inferred Bad	# Inferred Good
< 655	0	0	0%	0%	5	5	0
655 – 665	300	360	45.5%	54.5%	190	86	114
665 – 675	450	700	39.1%	60.9%	250	98	152
…	…	…	…	…	…	…	…

Parceling Augmentation – Example

```r
# Reject Inference - Parcelling #
parc <- seq(500, 725, 25)

accepts_scored$Score_parc <- cut(accepts_scored$Score, breaks = parc)
rejects_scored$Score_parc <- cut(rejects_scored$Score, breaks = parc)

table(accepts_scored$Score_parc, accepts_scored$bad)

parc_perc <- table(accepts_scored$Score_parc, accepts_scored$bad)[,2]/rowSums(table(accepts_scored$Score_parc, accepts_scored$bad))

rejects$bad <- 0

rej_bump <- 1.25

for(i in 1:(length(parc)-1)) {
  for(j in 1:length(rejects_scored$Score)) {
    if((rejects_scored$Score[j] > parc[i]) &
        rejects_scored$Score[j] < parc[i+1] &
        (runif(n = 1, min = 0, max = 1) < (rej_bump*parc_perc[i]))) {
      rejects$bad[j] <- 1
    }
  }
}

table(rejects_scored$Score_parc, rejects_scored$bad)

rejects$weight <- ifelse(rejects$bad == 1, 2.80, 0.59)
rejects$good <- abs(rejects$bad - 1)

comb_parc <- rbind(accepts, rejects) # New Combined Data Set #
```

Parceling Augmentation – Example

```
# Reject Inference - Parcelling #
parc <- seq(500, 725, 25)

accepts_scored$Score_parc <- cut(accepts_scored$Score, breaks = parc)
rejects_scored$Score_parc <- cut(rejects_scored$Score, breaks = parc)

table(accepts_scored$Score_parc, accepts_scored$bad)

parc_perc <- table(accepts_scored$Score_parc, accepts_scored$bad)[,2]/rowSums(table(accepts_scored$Score_parc, accepts_scored$bad))

rejects$bad <- 0

rej_bump <- 1.25

for(i in 1:(length(parc)-1)) {
  for(j in 1:length(rejects_scored$Score)) {
    if((rejects_scored$Score[j] > parc[i]) &
        rejects_scored$Score[j] < parc[i+1] &
        (runif(n = 1, min = 0, max = 1) < (rej_bump*parc_perc[i]))) {
      rejects$bad[j] <- 1
    }
  }
}

table(rejects_scored$Score_parc, rejects_scored$bad)

rejects$weight <- ifelse(rejects$bad == 1, 2.80, 0.59)
rejects$good <- abs(rejects$bad - 1)

comb_parc <- rbind(accepts, rejects) # New Combined Data Set #
```

Fuzzy Augmentation

1. Build a scorecard model using the known good/bad population (accepted applications)
2. Score the rejected applications with this model to obtain each rejected applicant's probability of being good, P(Good), and probability of being bad, P(Bad).
3. **Do not assign a reject to a good/bad class** – create **two weighted cases** for each rejected applicant using P(Good) and P(Bad).

Fuzzy Augmentation

4. Multiply P(Good) and P(Bad) by the user-specific rejection rate to form frequency variables.
5. For each rejected applicant, create **two observations** – one observation has a frequency variable (rejection weight × P(Good)) and a target variable of 0; other observation has a frequency variable (rejection weight × P(Bad)) and a target variable of 1.
6. Add inferred goods and bads back in with the known goods and bads and rebuild the scorecard.

Fuzzy Augmentation – Example

```
# Reject Inference - Fuzzy Augmentation #
rejects_scored$pred <- predict(initial_score, newdata=rejects_scored, type='response')

rejects_g <- rejects
rejects_b <- rejects

rejects_g$bad <- 0
rejects_g$weight <- (1-rejects_scored$pred)*2.80
rejects_g$good <- 1

rejects_b$bad <- 1
rejects_b$weight <- (rejects_scored$pred)*0.59
rejects_b$good <- 0

comb_fuzz <- rbind(accepts, rejects_g, rejects_b)
```

Reject Inference Techniques

- Three common techniques for reject inference:
  1. Hard Cutoff Augmentation
  2. Parceling Augmentation
  3. Fuzzy Augmentation (DEFAULT in SAS)

- There are other techniques as well, but are not as highly recommended.

Other Reject Inference Techniques

1. Assign all rejects to bads.
2. Assign rejects in the same proportion of goods to bads as reflected in the accepted data set.
3. Similar in-house model on different data.
4. Approve all applicants for certain period of time.
5. Clustering
6. Memory based reasoning

Other Reject Inference Techniques

1. Assign all rejects to bads
   - Appropriate only if approval rate is very high (ex. 97%) and there is a high degree of confidence in adjudication process.
2. Assign rejects in the same proportion of goods to bads as reflected in the accepted data set.
3. Similar in-house model on different data.
4. Approve all applicants for certain period of time.
5. Clustering
6. Memory based reasoning

Other Reject Inference Techniques

1. Assign all rejects to bads.
2. Assign rejects in the same proportion of goods to bads as reflected in the accepted data set.
   - Assignment done completely at random!
   - Valid only if current system has no consistency.
3. Similar in-house model on different data.
4. Approve all applicants for certain period of time.
5. Clustering
6. Memory based reasoning

Other Reject Inference Techniques

1. Assign all rejects to bads.
2. Assign rejects in the same proportion of goods to bads as reflected in the accepted data set.
3. Similar in-house model on different data.
   - Performance on similar products used as proxy.
   - Hard to pass by regulators.
4. Approve all applicants for certain period of time.
5. Clustering
6. Memory based reasoning

Other Reject Inference Techniques

1. Assign all rejects to bads.
2. Assign rejects in the same proportion of goods to bads as reflected in the accepted data set.
3. Similar in-house model on different data.
4. Approve all applicants for certain period of time.
   - Provides actual performance of rejects instead of inferred.
   - Might be “legal” problems…
5. Clustering
6. Memory based reasoning

Other Reject Inference Techniques

1. Assign all rejects to bads.
2. Assign rejects in the same proportion of goods to bads as reflected in the accepted data set.
3. Similar in-house model on different data.
4. Approve all applicants for certain period of time.
5. Clustering
6. Memory based reasoning

?

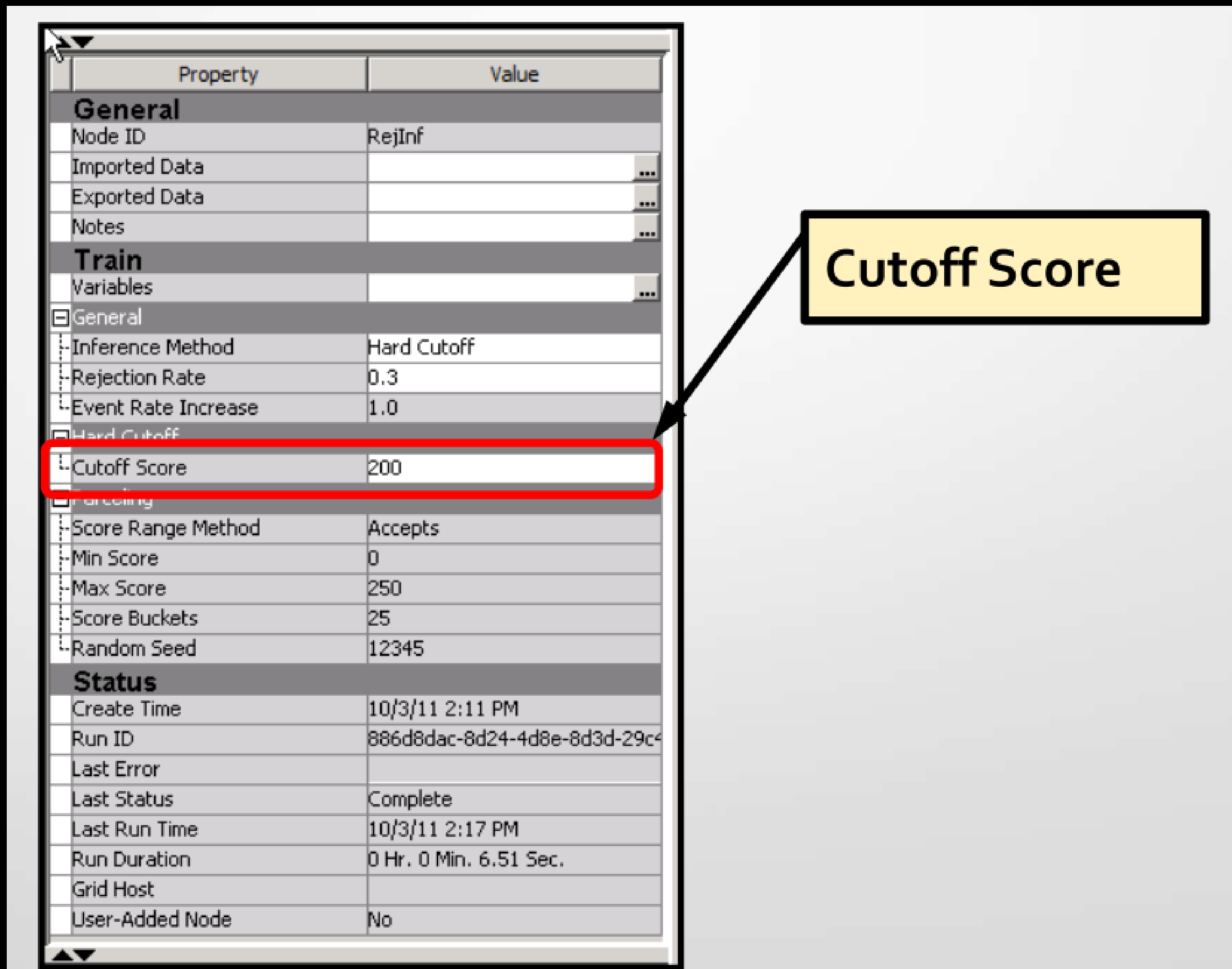
REJECT INFERENCE NODE IN SAS EM

General Options

Property	Value
General	
Node ID	RejInf
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
General	
Inference Method	Fuzzy
Rejection Rate	0.3
Event Rate Increase	1.0
Hard Cutoff	
Cutoff Score	200
Parceling	
Score Range Method	Accepts
Min Score	0
Max Score	250
Score Buckets	25
Random Seed	12345
Status	
Create Time	10/3/11 2:11 PM
Run ID	886d8dac-8d24-4d8e-8d3d-
Last Error	
Last Status	Complete
Last Run Time	10/3/11 2:17 PM
Run Duration	0 Hr. 0 Min. 6.51 Sec.
Grid Host	
User-Added Node	No

Inference Method
Rejection Rate
Event Rate Increase

Hard Cut-off Options



Property	Value
General	
Node ID	RejInf
Imported Data	
Exported Data	
Notes	
Train	
Variables	
General	
Inference Method	Hard Cutoff
Rejection Rate	0.3
Event Rate Increase	1.0
Hard Cutoff	
Cutoff Score	200
Score Range Method	Accepts
Min Score	0
Max Score	250
Score Buckets	25
Random Seed	12345
Status	
Create Time	10/3/11 2:11 PM
Run ID	886d8dac-8d24-4d8e-8d3d-29c
Last Error	
Last Status	Complete
Last Run Time	10/3/11 2:17 PM
Run Duration	0 Hr. 0 Min. 6.51 Sec.
Grid Host	
User-Added Node	No

Parceling Options

Property	Value
General	
Node ID	RejInf
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
General	
Inference Method	Parceling
Rejection Rate	0.3
Event Rate Increase	1.0
Hard Cutoff	
Cutoff Score	200
Parceling	
Score Range Method	Accepts
Min Score	0
Max Score	250
Score Buckets	25
Random Seed	12345
Status	
Create Time	10/3/11 2:11 PM
Run ID	886d8dac-8d24-4d8e-8d3d-29c4
Last Error	
Last Status	Complete
Last Run Time	10/3/11 2:17 PM
Run Duration	0 Hr. 0 Min. 6.51 Sec.
Grid Host	
User-Added Node	No

Score Range Method
Min Score
Max Score
Score Buckets
Random Seed

FINAL SCORECARD CREATION

Process Flow

Data Collection
- Variable Selection
- Sample Size
- Sample / Performance Window

Data Cleaning
- Eliminate Duplicates
- Examine / Remove Outliers

Variable Grouping and Selection
- Weights of Evidence (WOE)
- Information Value (IV)
- Gini Criterion

Initial Scorecard Creation
- Logistic Regression
- Accuracy
- Threshold
- Assessment

Reject Inference
- Remove bias resulting from exclusion of rejects

Final Scorecard Creation
- Final Model Assessment

Final Scorecard Creation

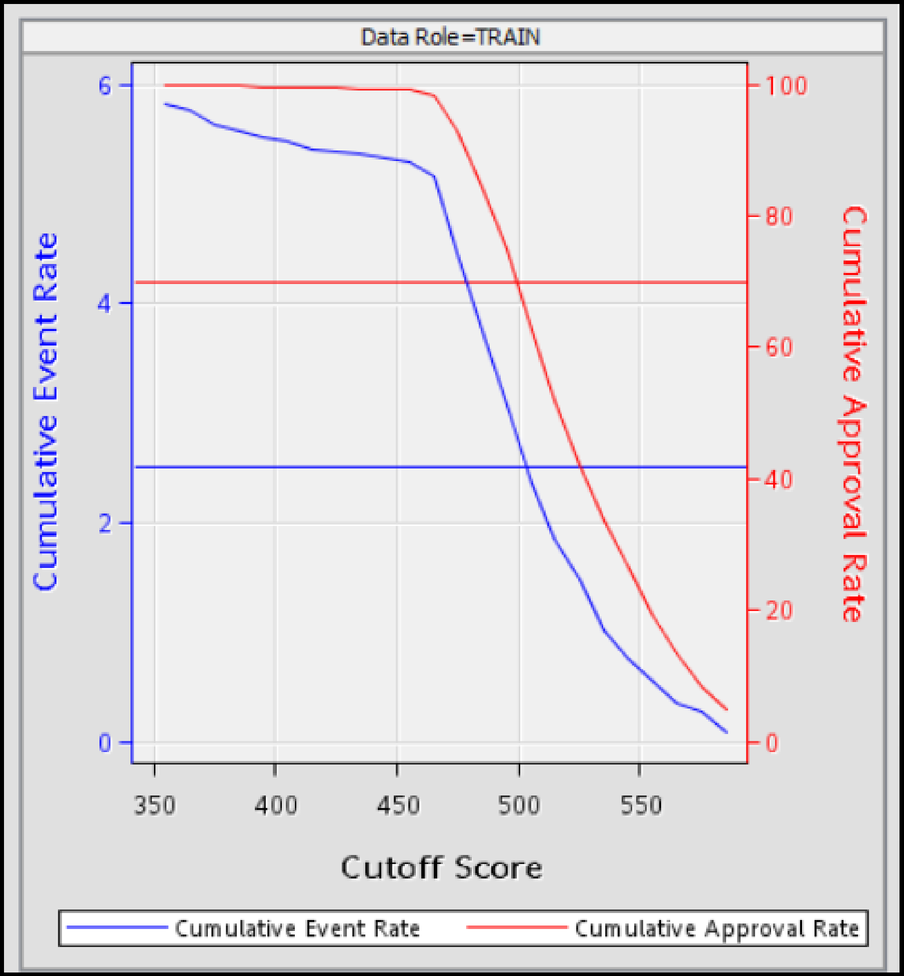
- The mechanics of building the final scorecard model are identical with the initial scorecard creation except that analysis is performed **after reject inference**.
- Accuracy Measurements:
  - Repeat review of the logistic model estimated parameters, life, KS, ROC, etc.

Defining Cut-off

- A new scored should be better than the last in terms of one of the following:
  - Lower bad rate for the same approval rate.
  - Higher approval rate while holding the bad rate constant.

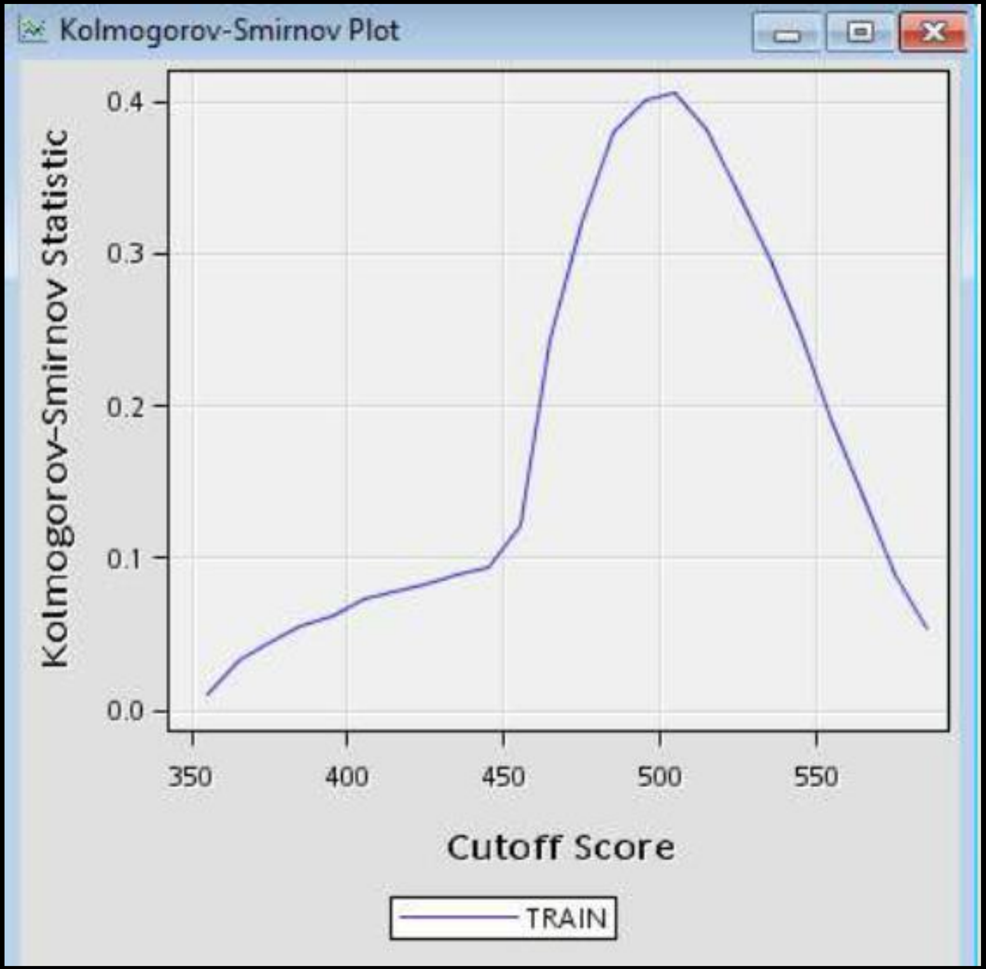
Defining Cut-off

- Trade-off Plots:
  - The reference lines of approval rate and event (bad) rate are predefined by analyst.
  - How much risk are you willing to take on?



Defining Cut-off

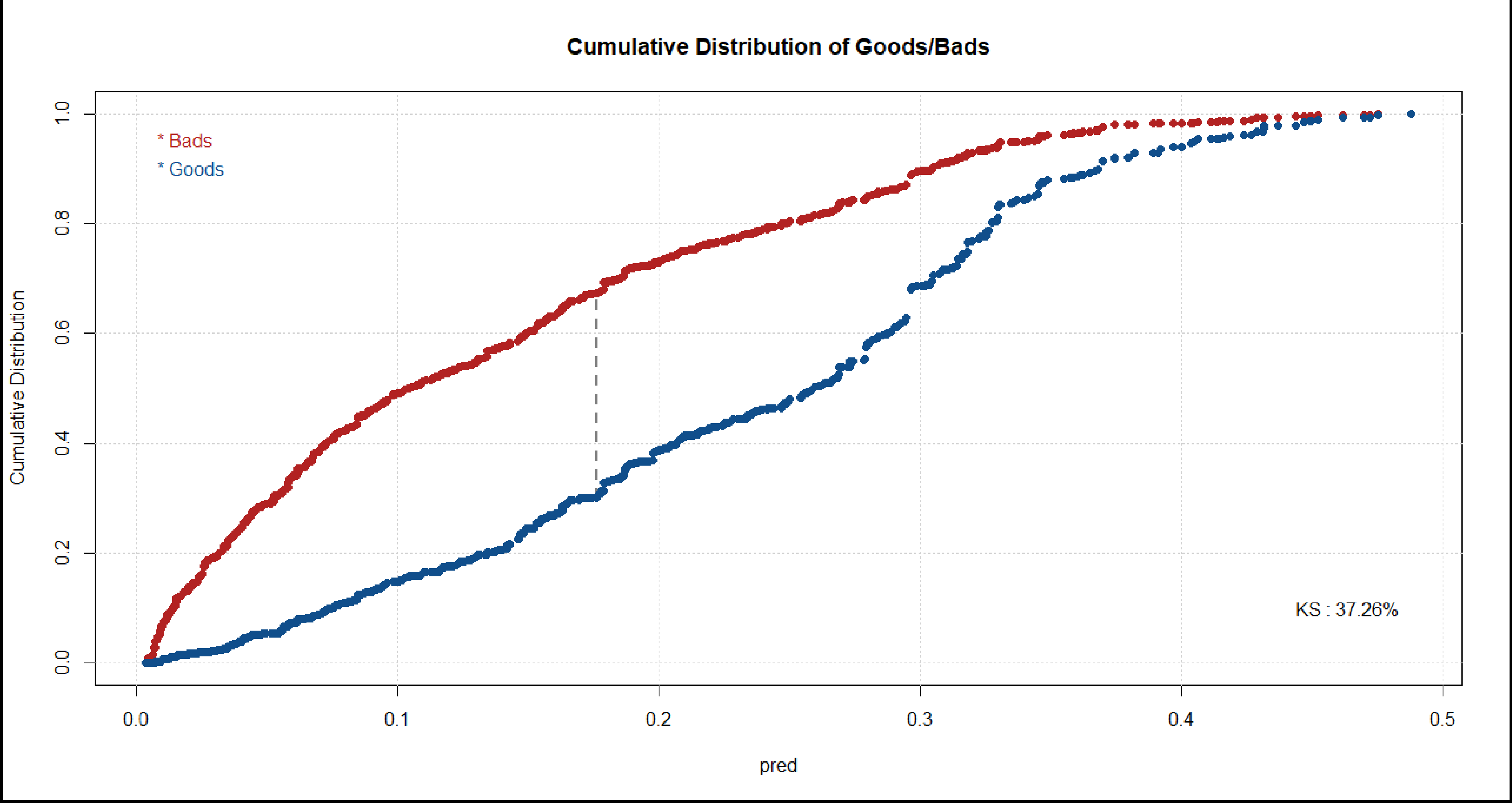
- KS Statistic Plot:
  - KS Statistic is how well your model separates the 1's and 0's – goods and bads.
  - Cut-off that maximizes the KS value.



Defining Cut-off

- Setting Multiple Cut-offs Example:
  - Anyone who scores above 210 points is accepted automatically.
  - Anyone who scores below 190 is declined.
  - Any scores in between 190 and 210 are referred to manual adjudication.

Final Scorecard – Example



Final Scorecard – Example