Model Selection and Cross Validation techniques

www.statinfer.com

Statinfer.com

Data Science Training and R&D

Training on
Data Science
Bigdata Analytics
Machine Learning
Predictive Modelling
Deep Learning

Corporate Training Online Video Courses	**Contact us** **info@statinfer.com** **+91-080 4851 1820** **+91-98867 60678**
Follow us on Social Media for more course material	Twitter: https://twitter.com/statinfer Facebook: https://www.facebook.com/statinfer/ Google Plus: https://plus.google.com/u/0/104559910764943078938 LinkedIn: https://www.linkedin.com/company/statinfer-software-solutions-llp

#647, 100 feet road, Indra Nagar, Bangalore, India

Note

- This presentation is just class notes. This course material is prepared by statinfer team, as an aid for training sessions.
- The best way to treat this is as a high-level summary; the actual session went more in depth and contained detailed information and examples
- Most of this material was written as informal notes, not intended for publication
- Please send questions/comments/corrections to info@statinfer.com
- Please check our website statinfer.com for latest version of this document

-Team Statinfer

Contents

Model Validation Metrics

Model Validation

- Checking how good is our model
- It is very important to report the accuracy of the model along with the final model
- The model validation in regression is done through R square and Adj R-Square
- Logistic Regression, Decision tree and other classification techniques have the very similar validation measures.
- Till now we have seen confusion matrix and accuracy. There are many more validation and model accuracy metrics for classification models

Classification-Validation measures

- Confusion matrix, Specificity, Sensitivity
- ROC, AUC
- Kappa, F1 Score
- KS, Gini
- Concordance and discordance
- Chi-Square, Hosmer and Lemeshow Goodness-of-Fit Test

All of them are measuring the model accuracy only. Some metrics work really well for certain class of problems. Confusion matrix, ROC and AUC will be sufficient for most of the business problems

statinfer

Sensitivity and Specificity

Classification Table

Sensitivity and Specificity are derived from confusion matrix

		Predicted Classes	
		0(Positive)	1(Negative)
Actual Classes	0(Positive)	True positive (TP) Actual condition is Positive, it is truly predicted as positive	False Negatives(FN) Actual condition is Positive, it is falsely predicted as negative
	1(Negative)	False Positives(FP) Actual condition is Negative, it is falsely predicted as positive	True Negatives(TN) Actual condition is Negative, it is truly predicted as negative

- Accuracy=(TP+TN)/(TP+FP+FN+TN)
- Misclassification Rate=(FP+FN)/(TP+FP+FN+TN)

Sensitivity and Specificity

- Sensitivity : Percentage of positives that are successfully classified as positive
- Specificity : Percentage of negatives that are successfully classified as negatives

Predicted Classes

Actual Classes

	0(Positive)	1(Negative)	
0(Positive)	True positive (TP) Actual condition is Positive, it is truly predicted as positive	False Negatives(FN) Actual condition is Positive, it is falsely predicted as negative	Sensitivity= TP/(TP+FN) or TP/ Overall Positives
1(Negative)	False Positives(FP) Actual condition is Negative, it is falsely predicted as positive	True Negatives(TN) Actual condition is Negative, it is truly predicted as negative	Specificity = TN/(TN+FP) or TN/ Overall Negatives

Sensitivity and Specificity

- By changing the threshold, the good and bad customers classification will be changed hence the sensitivity and specificity will be changed
- Which one of these two we should maximize? What should be ideal threshold?
- Ideally we want to maximize both Sensitivity & Specificity. But this is not possible always. There is always a tradeoff.
- Sometimes we want to be 100% sure on Predicted negatives, sometimes we want to be 100% sure on Predicted positives.
- Sometimes we simply don’t want to compromise on sensitivity sometimes we don’t want to compromise on specificity
- The threshold is set based on business problem

When Sensitivity is a high priority

When Sensitivity is a high priority

- Predicting a bad customers or defaulters before issuing the loan

Actual Classes	Predicted Classes: 0(Yes-Defaulter)	Predicted Classes: 1(Non-Defaulter)	
0(Yes-Defaulter)	True positive (TP) Actual customer is bad and model is predicting them as bad	False Negatives(FN) Actual customer is bad and model is predicting them as good	Sensitivity= TP/(TP+FN) or TP/ Overall Positives
1(Non-Defaulter)	False Positives(FP) Actual customer is good and model is predicting them as bad	True Negatives(TN) Actual customer is good and model is predicting them as good	Specificity = TN/(TN+FP) or TN/ Overall Negatives

When Sensitivity is a high priority

- Predicting a bad defaulters before issuing the loan

Predicted Classes

Actual Classes

	0(Yes-Defaulter)	1(Non-Defaulter)	
0(Yes-Defaulter)	True positive (TP) Actual customer is bad and model is predicting them as bad. **Rejected a Loan of 100,000**	False Negatives(FN) Actual customer is bad and model is predicting them as good **Issued a loan of 100,00**	Sensitivity= TP/(TP+FN) or TP/ Overall Positives
1(Non-Defaulter)	False Positives(FP) Actual customer is good and model is predicting them as bad. **Rejected a Loan of 100,000**	True Negatives(TN) Actual customer is good and model is predicting them as good. **Issued a loan of 100,00**	Specificity = TN/(TN+FP) or TN/ Overall Negatives

When Sensitivity is a high priority

- The profit on good customer loan is not equal to the loss on one bad customer loan
- The loss on one bad loan might eat up the profit on 100 good customers
- In this case one bad customer is not equal to one good customer.
- If p is probability of default then we would like to set our threshold in such a way that we don’t miss any of the bad customers.
- We set the threshold in such a way that Sensitivity is high
- We can compromise on specificity here. If we wrongly reject a good customer, our loss is very less compared to giving a loan to a bad customer.
- We don’t really worry about the good customers here, they are not harmful hence we can have less Specificity

statinfer

When Specificity is a high priority

When Specificity is a high priority

- Testing a medicine is good or poisonous

		Predicted Classes		
		0(Yes-Good)	1(Poisonous)	
Actual Classes	0(Yes-Good)	True positive (TP) Actual medicine is good and model is predicting them as good	False Negatives(FN) Actual medicine is good and model is predicting them as poisonous	Sensitivity= TP/(TP+FN) or TP/ Overall Positives
	1(Poisonous)	False Positives(FP) Actual medicine is poisonous and model is predicting them as good	True Negatives(TN) Actual medicine is poisonous and model is predicting them as poisonous	Specificity = TN/(TN+FP) or TN/ Overall Negatives

When Specificity is a high priority

- Testing a medicine is good or poisonous

		Predicted Classes		
		0(Yes-Good)	1(Poisonous)	
Actual Classes	0(Yes-Good)	True positive (TP) Actual medicine is good and model is predicting them as good. **Recommended for use**	False Negatives(FN) Actual medicine is good and model is predicting them as poisonous. **Banned the usage**	Sensitivity= TP/(TP+FN) or TP/ Overall Positives
	1(Poisonous)	False Positives(FP) Actual medicine is poisonous and model is predicting them as good. **Recommended for use**	True Negatives(TN) Actual medicine is poisonous and model is predicting them as poisonous. **Banned the usage**	Specificity = TN/(TN+FP) or TN/ Overall Negatives

When Specificity is a high priority

- In this case, we have to really avoid cases like , Actual medicine is poisonous and model is predicting them as good.
- We can’t take any chance here.
- The specificity need to be near 100.
- The sensitivity can be compromised here. It is not very harmful not to use a good medicine when compared with vice versa case

Sensitivity vs Specificity - Importance

- There are some cases where Sensitivity is important and need to be near to 1
- There are business cases where Specificity is important and need to be near to 1
- We need to understand the business problem and decide the importance of Sensitivity and Specificity

Calculating Sensitivity and Specificity

LAB - Sensitivity and Specificity

- Build a logistic regression model on fiber bits data
- Create the confusion matrix
- Find the accuracy
- Calculate Specificity
- Calculate Sensitivity

Code - Sensitivity and Specificity

```
> Fiberbits_model_1<-glm(active_cust~.,family=binomial(),data=Fiberbits)
Warning message:
glm.fit: fitted probabilities numerically 0 or 1 occurred
> summary(Fiberbits_model_1)

Call:
glm(formula = active_cust ~ ., family = binomial(), data = Fiberbits)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-8.4904  -0.8752   0.4055   0.7619   2.9465

Coefficients:
                             Estimate Std. Error z value Pr(>|z|)
(Intercept)                -1.761e+01  3.008e-01  -58.54   <2e-16 ***
income                      1.710e-03  8.213e-05   20.82   <2e-16 ***
months_on_network           2.880e-02  1.005e-03   28.65   <2e-16 ***
Num_complaints             -6.865e-01  3.010e-02  -22.81   <2e-16 ***
number_plan_changes        -1.896e-01  7.603e-03  -24.94   <2e-16 ***
relocated                  -3.163e+00  3.957e-02  -79.93   <2e-16 ***
monthly_bill               -2.198e-03  1.571e-04  -13.99   <2e-16 ***
technical_issues_per_month -3.904e-01  7.152e-03  -54.58   <2e-16 ***
Speed_test_result           2.222e-01  2.378e-03   93.44   <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: 136149  on 99999  degrees of freedom
Residual deviance:  98359  on 99991  degrees of freedom
AIC: 98377

Number of Fisher Scoring iterations: 8
```

Code - Sensitivity and Specificity (threshold=0.5)

```
> threshold=0.5
> predicted_values<-ifelse(predict(Fiberbits_model_1,type="response")>threshold,1,0)
> table(predicted_values)
predicted_values
    0     1
40339 59661
>
> actual_values<-Fiberbits_model_1$y
> conf_matrix<-table(actual_values,predicted_values)
> conf_matrix
             predicted_values
actual_values     0     1
            0 29492 12649
            1 10847 47012
>
> sensitivity=conf_matrix[1,1]/(conf_matrix[1,1]+conf_matrix[1,2])
> print(sensitivity)
[1] 0.699841
> specificity=conf_matrix[2,2]/(conf_matrix[2,1]+conf_matrix[2,2])
> print(specificity)
[1] 0.812527
```

Code – Change the threshold to 0.8

```
> threshold=0.8
> predicted_values<-ifelse(predict(Fiberbits_model_1,type="response")>threshold,1,0)
> table(predicted_values)
predicted_values
    0     1
68288 31712
>
> actual_values<-Fiberbits_model_1$y
> conf_matrix<-table(actual_values,predicted_values)
> conf_matrix
             predicted_values
actual_values     0     1
            0 37767  4374
            1 30521 27338
>
> sensitivity=conf_matrix[1,1]/(conf_matrix[1,1]+conf_matrix[1,2])
> print(sensitivity)
[1] 0.8962056
> specificity=conf_matrix[2,2]/(conf_matrix[2,1]+conf_matrix[2,2])
> print(specificity)
[1] 0.4724935
```

Code – Change the threshold to 0.3

```
> threshold=0.3
> predicted_values<-ifelse(predict(Fiberbits_model_1,type="response")>threshold,1,0)
> table(predicted_values)
predicted_values
    0     1 
17205 82795 
> 
> actual_values<-Fiberbits_model_1$y
> conf_matrix<-table(actual_values,predicted_values)
> conf_matrix
             predicted_values
actual_values     0     1
            0 15832 26309
            1  1373 56486
> 
> sensitivity=conf_matrix[1,1]/(conf_matrix[1,1]+conf_matrix[1,2])
> print(sensitivity)
[1] 0.3756911
> specificity=conf_matrix[2,2]/(conf_matrix[2,1]+conf_matrix[2,2])
> print(specificity)
[1] 0.9762699
```

statinfer

ROC Curve

ROC Curve

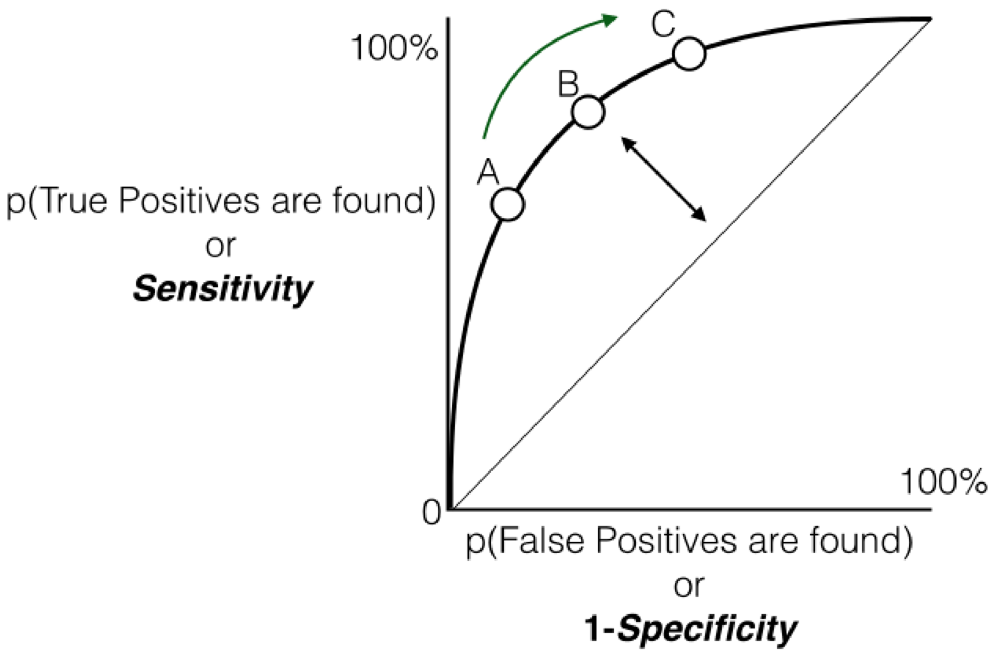
- Each threshold gives a sensitivity and specificity pair.
- What is the optimal sensitivity and specificity for a given problem?
- ROC curves helps us in choosing optimal sensitivity and specificity pair.
- ROC tells us, how many mistakes are we making to identify all the positives?

ROC Curve

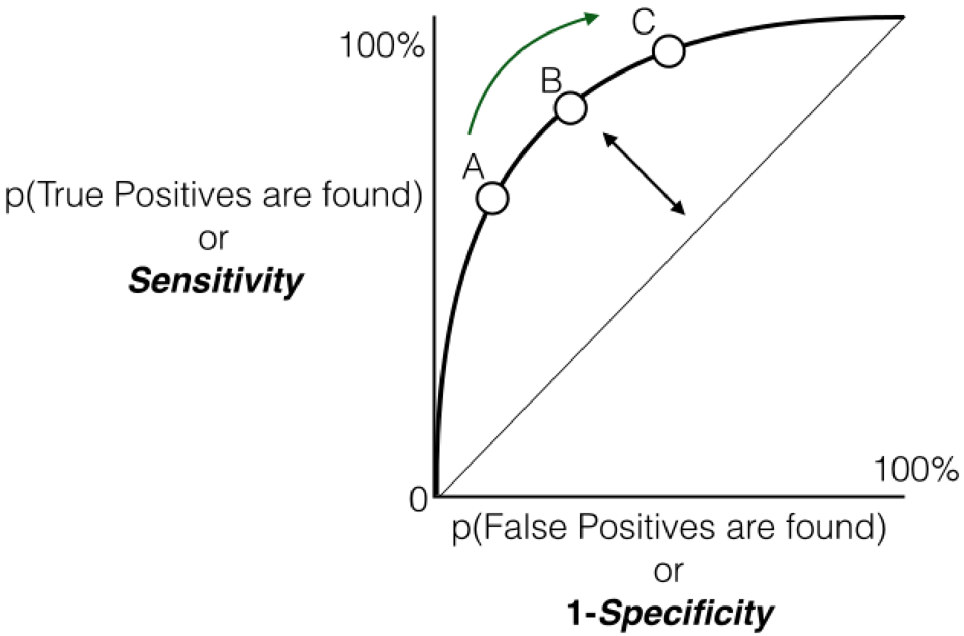
- ROCROC(Receiver operating characteristic) tells us, how many mistakes are we making to identify all the positives?

- ROC tells us, how many mistakes(False positives) are we making to identify all the positives?

- Curve is drawn by taking False positive rate on X-axis and True positive rate on Y- axis



ROC Curve - Interpretation



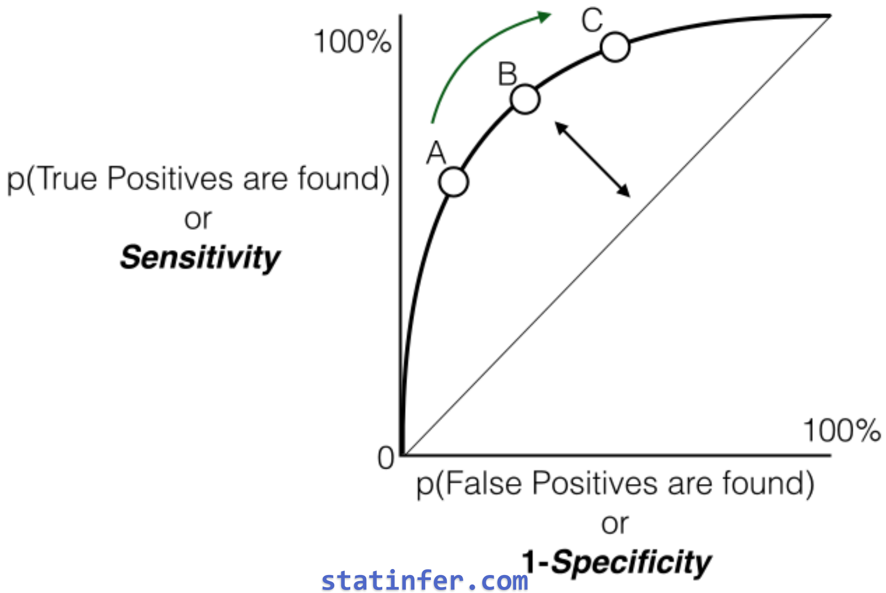
- How many mistakes are we making to identify all the positives?
- How many mistakes are we making to identify 70%, 80% and 90% of positives?
- 1-Specificty(false positive rate) gives us an idea on mistakes that we are making
- We would like to make 0% mistakes for identifying 100% positives
- We would like to make very minimal mistakes for identifying maximum positives
- We want that curve to be far away from straight line
- Ideally we want the area under the curve as high as possible

AUC

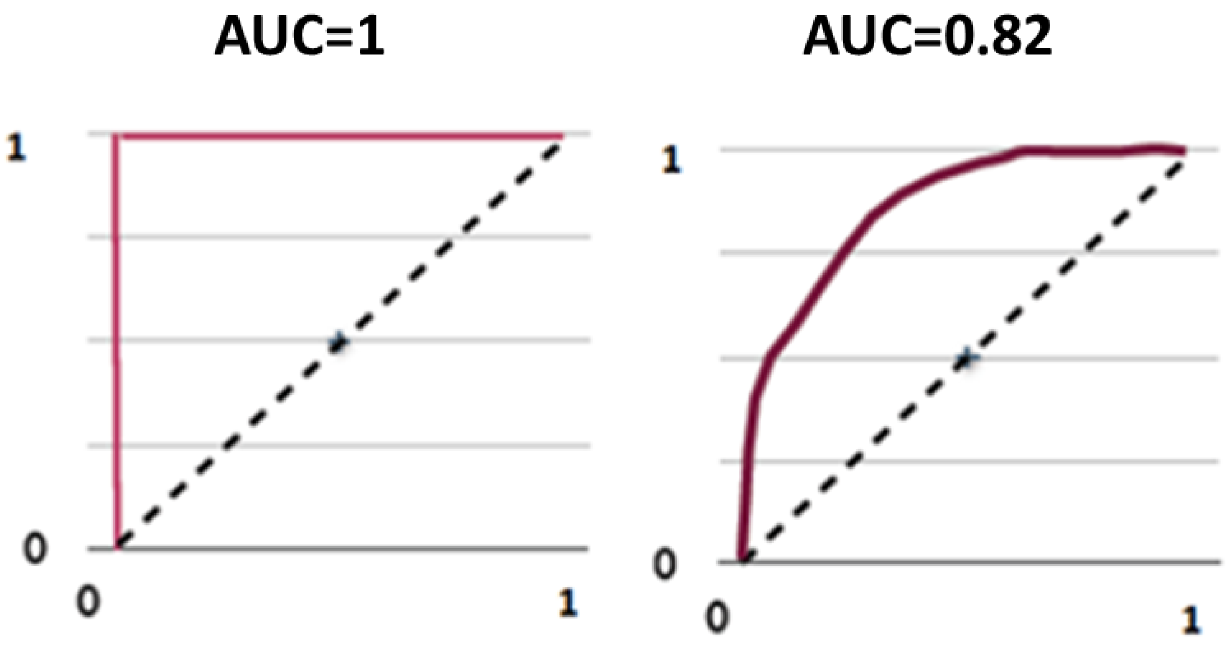
statinfer

ROC and AUC

- We want that curve to be far away from straight line. Ideally we want the area under the curve as high as possible
- ROC comes with a connected topic, AUC. Area Under
- ROC Curve Gives us an idea on the performance of the model under all possible values of threshold.
- We want to make almost 0% mistakes while identifying all the positives, which means we want to see AUC value near to 1



AUC



- AUC is near to 1 for a good model

statinfer

ROC and AUC Calculation

LAB: ROC and AUC

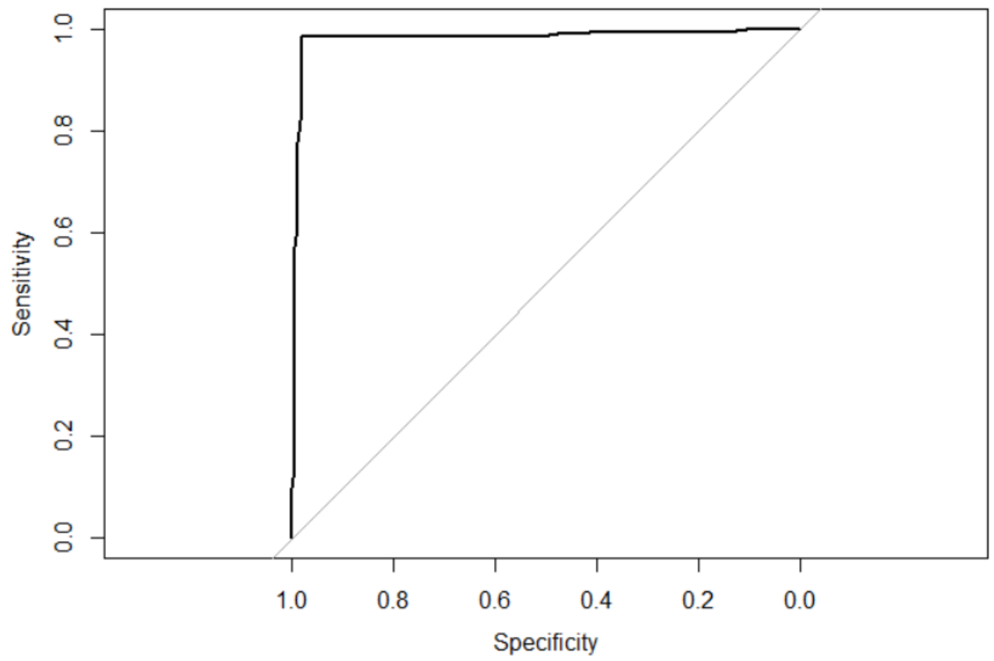
- Calculate ROC and AUC for Product Sales Data/Product_sales.csv logistic regression model
- Calculate ROC and AUC for fiber bits logistic regression model

Code: ROC and AUC

```
> #For product Sales model
> Product_sales <- read.csv("Product Sales Data/Product_sales.csv")
> names(Product_sales)
[1] "Age"    "Bought"
>
> prod_sales_Logit_model <- glm(Bought ~ Age,family=binomial(),data=Product_sales)
> summary(prod_sales_Logit_model)

Call:
glm(formula = Bought ~ Age, family = binomial(), data = Product_sales)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6922  -0.1645  -0.0619   0.1246   3.5378

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.90975    0.72755  -9.497   <2e-16 ***
Age          0.21786    0.02091  10.418   <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: 640.425  on 466  degrees of freedom
Residual deviance:  95.015  on 465  degrees of freedom
AIC: 99.015

Number of Fisher Scoring iterations: 7
```

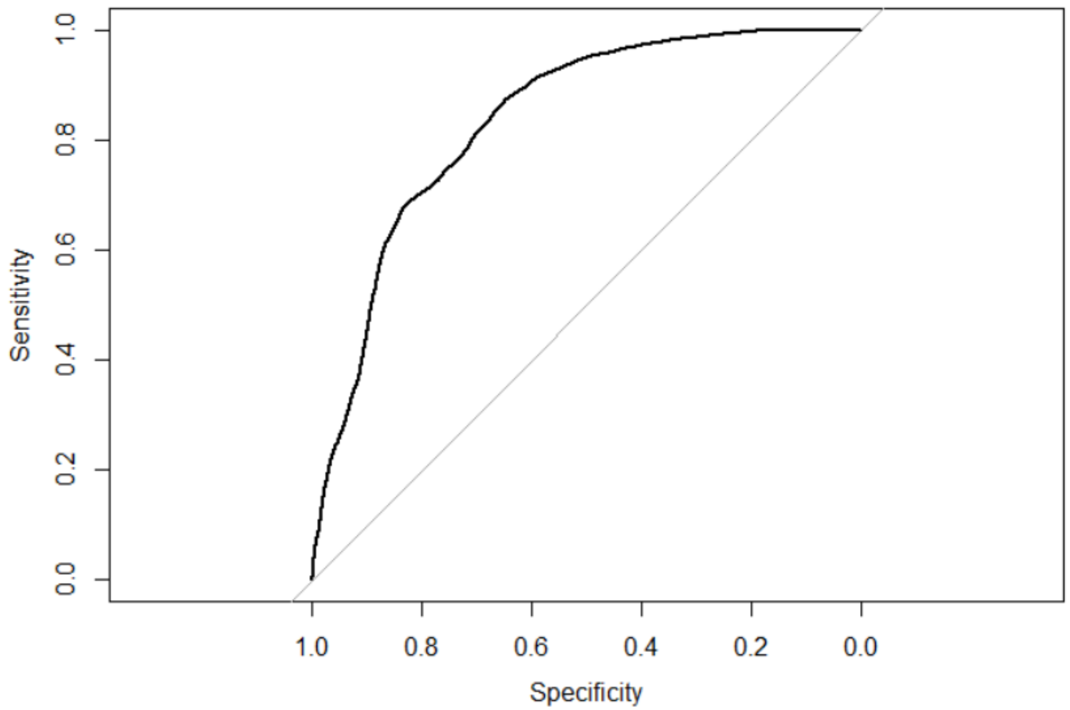
Code: ROC and AUC

```
> library(pROC)
> predicted_prob<-predict(prod_sales_Logit_model,type="response")
> roccurve <- roc(prod_sales_Logit_model$y, predicted_prob)
> plot(roccurve)

Call:
roc.default(response = prod_sales_Logit_model$y, predictor = predicted_prob)

Data: predicted_prob in 262 controls (prod_sales_Logit_model$y 0) < 205 cases (prod_sales_Logit_mo
del$y 1).
Area under the curve: 0.983
>
> auc(roccurve)
Area under the curve: 0.983
> auc(prod_sales_Logit_model$y, predicted_prob)
Area under the curve: 0.983
```



Code: ROC and AUC

```
> #For Fiber bits model
> predicted_prob<-predict(Fiberbits_model_1,type="response")
> roccurve <- roc(Fiberbits_model_1$y, predicted_prob)
> plot(roccurve)

Call:
roc.default(response = Fiberbits_model_1$y, predictor = predicted_prob)

Data: predicted_prob in 42141 controls (Fiberbits_model_1$y 0) < 57859 cases (Fiberbits_model_1$y
1).
Area under the curve: 0.835
>
> auc(roccurve)
Area under the curve: 0.835
> auc(Fiberbits_model_1$y, predicted_prob)
Area under the curve: 0.835
```



When to use AUC over Accuracy?

- AUC is not same as accuracy. Accuracy is calculated at one cut-off point.
- Use AUC when you want to work with probabilities and scoring rather than simply classifying on one threshold
- Use AUC when each point probability is important for you than accuracy on two classes.
- Use AUC in case of class imbalance. When one false positive misclassification is not same as on negative misclassification

statinfer

The best model

What is a best model? How to build?

- A model with maximum accuracy /least error
- A model that uses maximum information available in the given data
- A model that has minimum squared error
- A model that captures all the hidden patterns in the data
- A model that produces the best perdition results

Model Selection

- How to build/choose a best model?
- Error on the training data is not a good meter of performance on future data
- How to select the best model out of the set of available models ?
- Are there any methods/metrics to choose best model?
- What is training error? What is testing error? What is hold out sample error?

LAB: The most accurate model

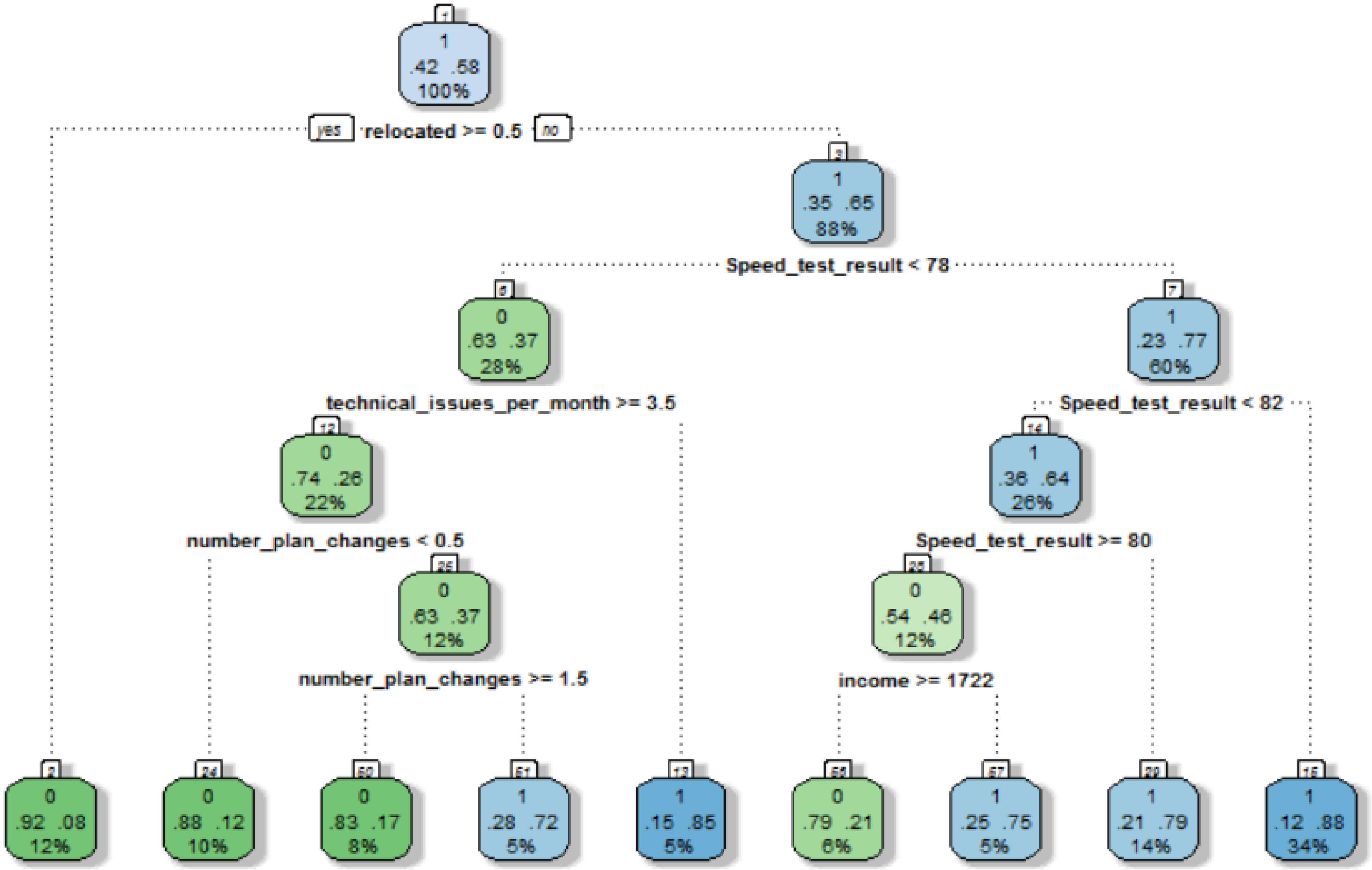
LAB: The most accurate model

- Data: Fiberbits/Fiberbits.csv
- Build a decision tree to predict active_user
- What is the accuracy of your model?
- Grow the tree as much as you can and achieve 95% accuracy.

Code: The most accurate model

```
> ###Model1
> library(rpart)
> Fiber_bits_tree1<-rpart(active_cust~., method="class", control=rpart.control(minsplit=30, cp=0
.01), data=Fiberbits)
>
> library(rattle)
> fancyRpartPlot(Fiber_bits_tree1)
>
> Fbits_pred1<-predict(Fiber_bits_tree1, type="class")
> conf_matrix1<-table(Fbits_pred1,Fiberbits$active_cust)
> conf_matrix1

Fbits_pred1     0     1
          0 31513  4743
          1 10628 53116
> accuracy1<-(conf_matrix1[1,1]+conf_matrix1[2,2])/(sum(conf_matrix1))
> accuracy1
[1] 0.84629
```

Code: The most accurate model



Code: The most accurate model

```
> ###Model2
> Fiber_bits_tree2<-rpart(active_cust~., method="class", control=rpart.control(minsplit=5, cp
=0.000001), data=Fiberbits)
> Fbits_pred2<-predict(Fiber_bits_tree2, type="class")
> conf_matrix2<-table(Fbits_pred2,Fiberbits$active_cust)
> conf_matrix2

Fbits_pred2     0     1
          0 39363  2159
          1  2778 55700
> accuracy2<-(conf_matrix2[1,1]+conf_matrix2[2,2])/(sum(conf_matrix2))
> accuracy2
[1] 0.95063
```

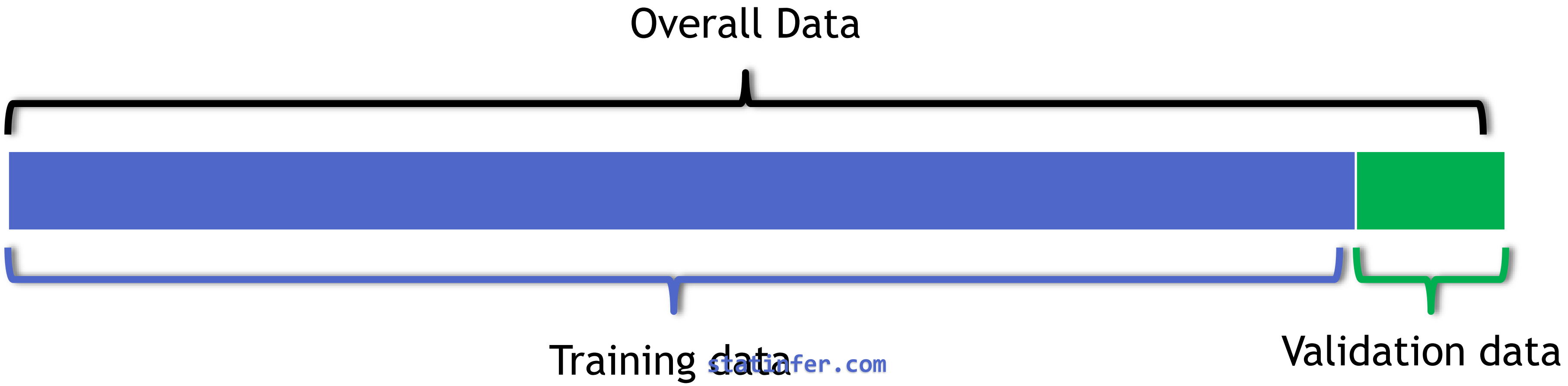
Different type of datasets and errors

The Training Error

- The accuracy of our best model is 95%. Is the 5% error model really good?
- The error on the training data is known as training error.
- A low error rate on training data may not always mean the model is good.
- What really matters is how the model is going to perform on unknown data or test data.
- We need to find out a way to get an idea on error rate of test data.
- We may have to keep aside a part of the data and use it for validation.
- There are two types of datasets and two types of errors

Two types of datasets

- There are two types of datasets
  - **Training set:** This is used in model building. The input data
  - **Test set:** The unknown dataset. This dataset is gives the accuracy of the final model
- We may not have access to these two datasets for all machine learning problems. In some cases, we can take 90% of the available data and use it as training data and rest 10% can be treated as validation data
    - **Validation set:** This dataset kept aside for model validation and selection. This is a temporary subsite to test dataset. It is not third type of data
- We create the validation data with the hope that the error rate on validation data will give us some basic idea on the test error



Types of errors

- The training error
  - The error on training dataset
  - In-time error
  - Error on the known data
  - Can be reduced while building the model
- The test error
  - The error that matters
  - Out-of-time error
  - The error on unknown/new dataset.

"A good model will have both training and test error very near to each other and close to zero"

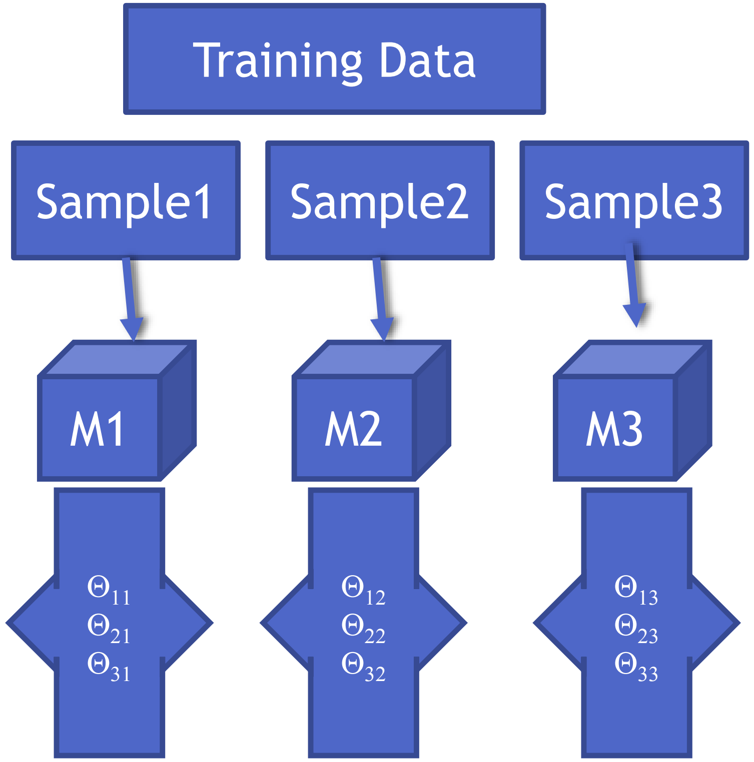
The problem of over fitting

The problem of over fitting

- In search of the best model on the given data we add many predictors, polynomial terms, Interaction terms, variable transformations, derived variables, indicator/dummy variables etc.,
- Most of the times we succeed in reducing the error. What error is this?
- So by complicating the model we fit the best model for the training data.
- Sometimes the error on the training data can reduce to near zero
- But the same best model on training data fails miserably on test data.

The problem of over fitting

- Imagine building multiple models with small changes in training data.
- The resultant set of models will have huge variance in their parameter estimates.
- If we build a model on sample1 of training data.
  - Sample2 will almost have the same properties as sample1 but the coefficients in the model change drastically if the model is over fitted.
  - Same with case of training sample3
- Hence over fitted models are the models with huge variance (variance in model parameters)
- The model is made really complicated, that it is very sensitive to minimal changes
- In simple terms –Variance is how much the model parameters changes with small changes in training data



The problem of over fitting

- By complicating the model the variance of the parameters estimates inflates
- Model tries to fit the irrelevant characteristics in the data
- Over fitting
  - The model is super good on training data but not so good on test data
  - We fit the model for the noise in the data
  - Less training error, high testing error
  - The model is over complicated with too many predictors
  - Model need to be simplified
  - A model with lot of **variance**

LAB: Model with huge Variance

LAB: Model with huge Variance

- Data: Fiberbits/Fiberbits.csv
- Take initial 90% of the data. Consider it as training data. Keep the final 10% of the records for validation.
- Build the best model(5% error) model on training data.
- Use the validation data to verify the error rate. Is the error rate on the training data and validation data same?

LAB: Model with huge Variance

```
> #########Overfitting
> ###Model on training data
> library(rpart)
> Fiber_bits_tree3<-rpart(active_cust~., method="class", control=rpart.control(minsplit=5, cp
=0.000001), data=fiber_bits_train)
> Fbits_pred3<-predict(Fiber_bits_tree3, type="class")
> conf_matrix3<-table(Fbits_pred3,fiber_bits_train$active_cust)
> conf_matrix3

Fbits_pred3     0     1
          0 33769  1832
          1  2444 51955
> accuracy3<-(conf_matrix3[1,1]+conf_matrix3[2,2])/(sum(conf_matrix3))
> accuracy3
[1] 0.9524889
>
> ###Validation accuracy
> fiber_bits_validation$pred <- predict(Fiber_bits_tree3, fiber_bits_validation,type="class")
> conf_matrix_val<-table(fiber_bits_validation$pred,fiber_bits_validation$active_cust)
> conf_matrix_val

       0    1
  0 3504  460
  1 2424 3612
> accuracy_val<-(conf_matrix_val[1,1]+conf_matrix_val[2,2])/(sum(conf_matrix_val))
> accuracy_val
[1] 0.7116
```

The problem of under fitting

The problem of under-fitting

- Simple models are better. Its true but is that always true? May not be always true.
- We might have given it up too early. Did we really capture all the information?
- Did we do enough research and future reengineering to fit the best model? Is it the best model that can be fit on this data?
- By being over cautious about variance in the parameters, we might miss out on some patterns in the data.
- Model need to be complicated enough to capture all the information present.

The problem of under-fitting

- If the training error itself is high, how can we be so sure about the model performance on unknown data?
- Most of the accuracy and error measuring statistics give us a clear idea on training error, this is one advantage of under fitting, we can identify it confidently.
- Under fitting
  - A model that is too simple
  - A mode with a scope for improvement
  - A model with lot of **bias**

LAB: Model with huge Bias

LAB: Model with huge Bias

- Lets simplify the model.
- Take the high variance model and prune it.
- Make it as simple as possible.
- Find the training error and validation error.

LAB: Model with huge Bias

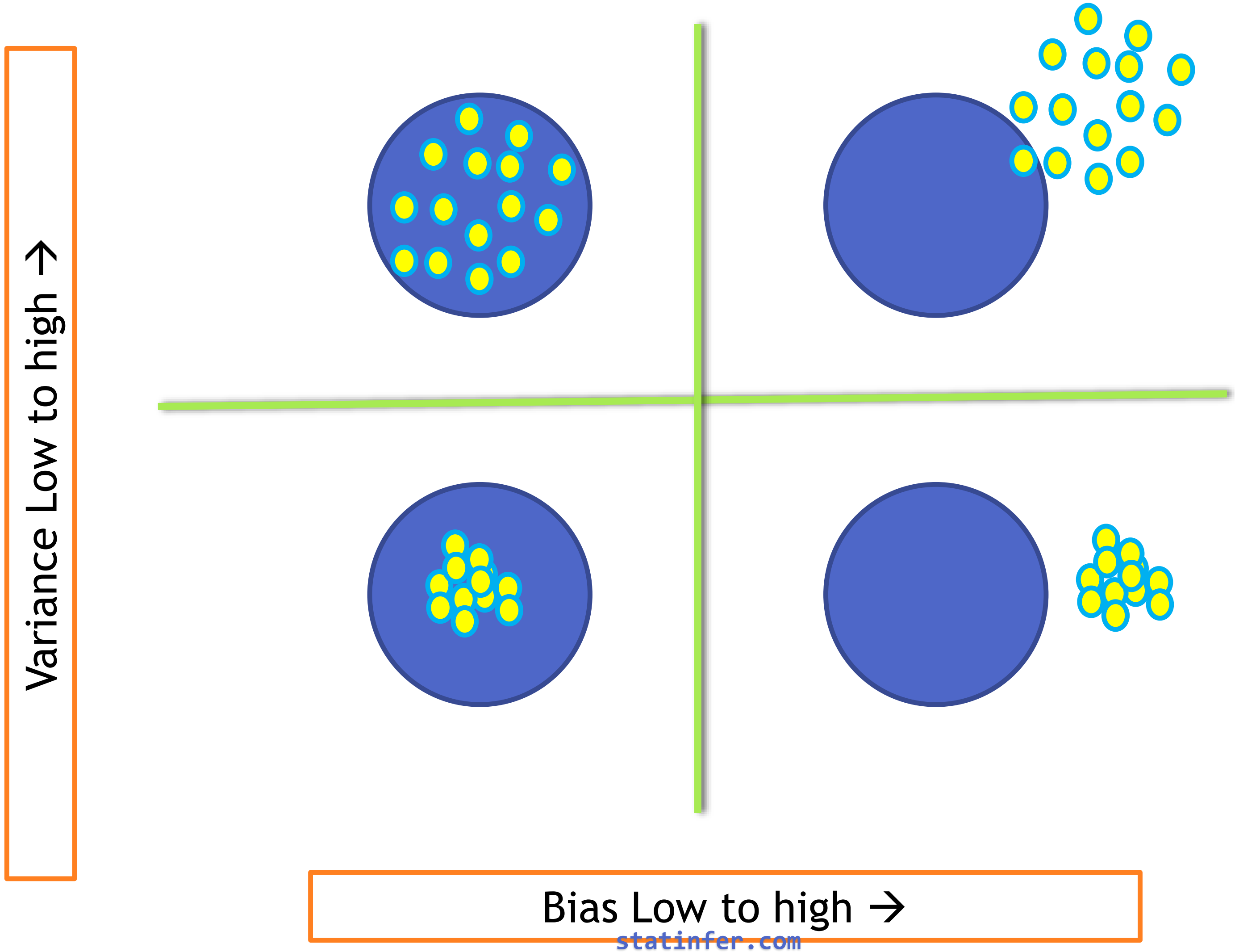
```
> #########Underfitting
> ###Simple Model
> Fiber_bits_tree4<-rpart(active_cust~., method="class", control=rpart.control(minsplit=30, c
p=0.25), data=fiber_bits_train)
>
> library(rattle)
> fancyRpartPlot(Fiber_bits_tree4)
>
> Fbits_pred4<-predict(Fiber_bits_tree4, type="class")
> conf_matrix4<-table(Fbits_pred4,fiber_bits_train$active_cust)
> conf_matrix4

Fbits_pred4     0     1
          0 11209   921
          1 25004 52866
> accuracy4<-(conf_matrix4[1,1]+conf_matrix4[2,2])/(sum(conf_matrix4))
> accuracy4
[1] 0.7119444
>
> ###Validation accuracy
> fiber_bits_validation$pred1 <- predict(Fiber_bits_tree4, fiber_bits_validation,type="class"
)
> conf_matrix_val1<-table(fiber_bits_validation$pred1,fiber_bits_validation$active_cust)
> conf_matrix_val1

       0    1
  0  185   33
  1 5743 4039
> accuracy_val1<-(conf_matrix_val1[1,1]+conf_matrix_val1[2,2])/(sum(conf_matrix_val1))
> accuracy_val1
[1] 0.4224
```

Model Bias and Variance

Model Bias and Variance

- Over fitting
  - Low Bias with High Variance
  - Low training error – 'Low Bias'
  - High testing error
  - Unstable model – 'High Variance'
  - The coefficients of the model change with small changes in the data
- Under fitting
  - High Bias with low Variance
  - High training error – 'high Bias'
  - testing error almost equal to training error
  - Stable model – 'Low Variance'
  - The coefficients of the model doesn't change with small changes in the data

Model Bias and Variance



Model aim is to hit the center of circle

statinfer

The Bias-Variance Decomposition

$$Y = f(X) + \varepsilon$$

$$Var(\varepsilon) = \sigma^2$$

$$\begin{aligned} SquaredError &= E[(Y - \hat{f}(x_0))^2 \mid X = x_0] \\ &= \sigma^2 + [E\hat{f}(x_0) - f(x_0)]^2 + E[\hat{f}(x_0) - E\hat{f}(x_0)]^2 \\ &= \sigma^2 + Bias^2(\hat{f}(x_0)) + Var(\hat{f}(x_0)) \end{aligned}$$

Overall Model Squared Error = Irreducible Error + Bias^2 + Variance

Bias-Variance Decomposition

- Overall Model Squared Error = Irreducible Error + Bias^2 + Variance
- Overall error is made by bias and variance together
- High bias low variance, Low bias and high variance, both are bad for the overall accuracy of the model
- A good model need to have low bias and low variance or at least an optimal where both of them are jointly low
- How to choose such optimal model. How to choose that optimal model complexity

statinfer

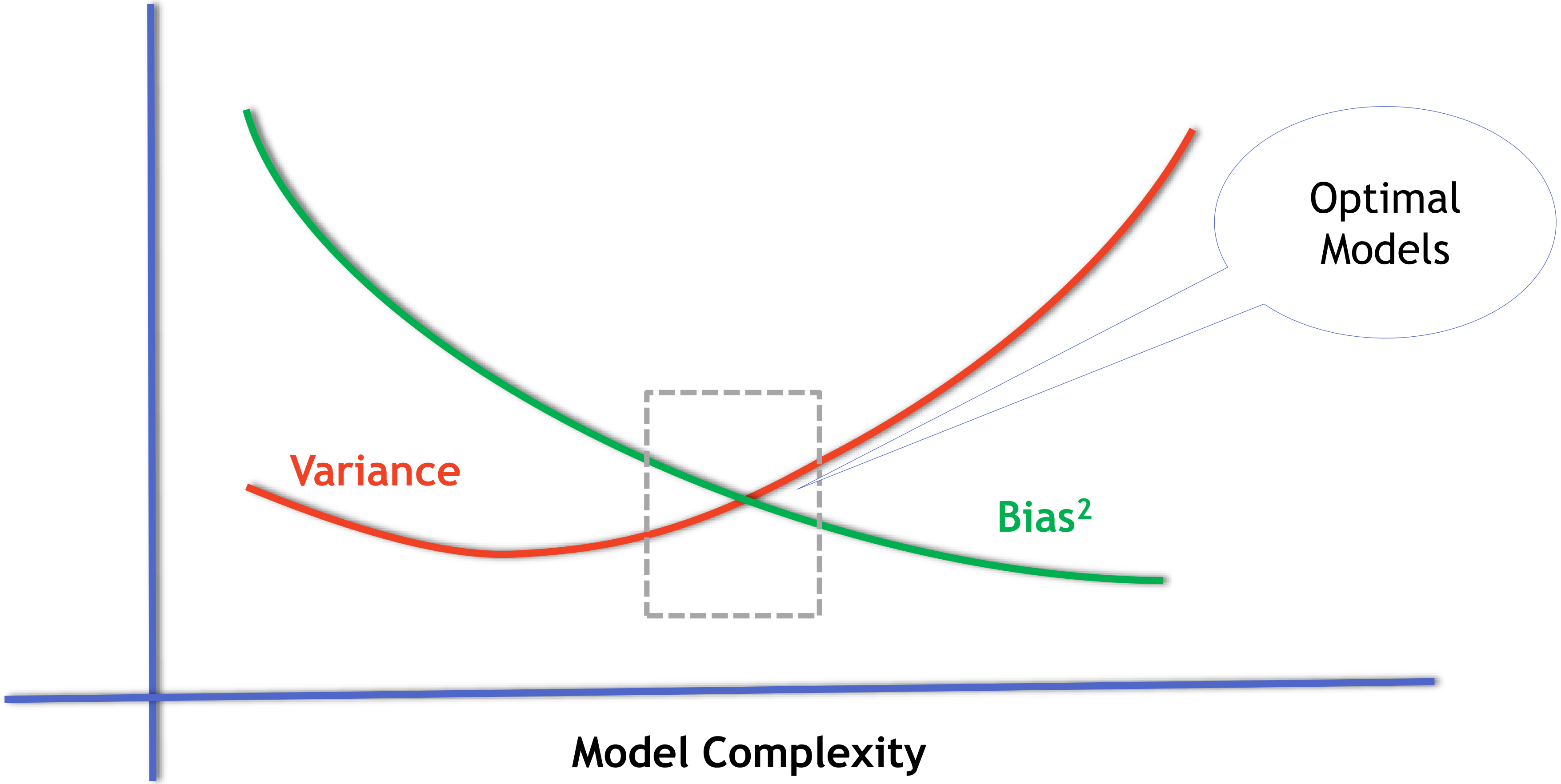
Choosing optimal model-Bias Variance Tradeoff

Two ways of reading bias and variance

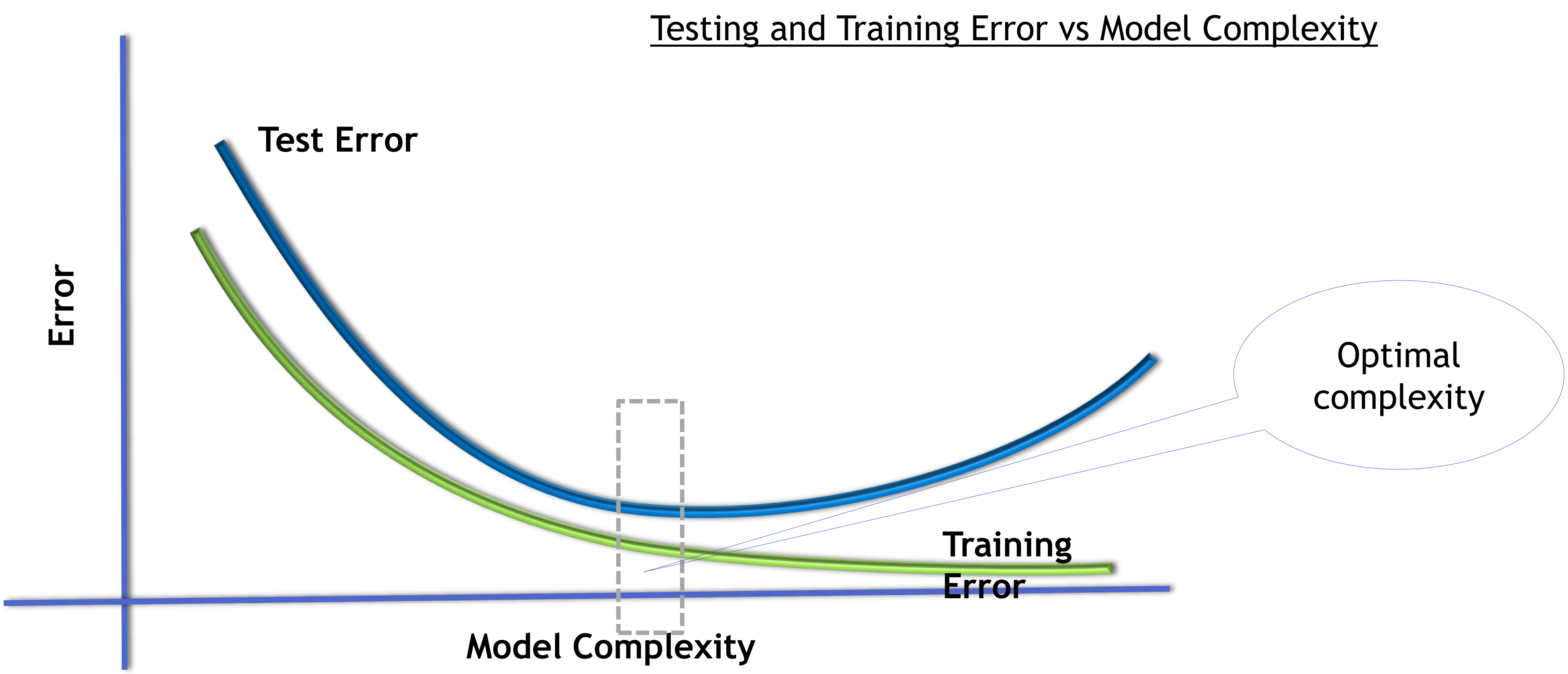
- Variance and bias vs Model Complexity
- Testing and Training Error vs Model Complexity

Bias Variance Tradeoff

Variance and bias vs Model Complexity



Test and Training error



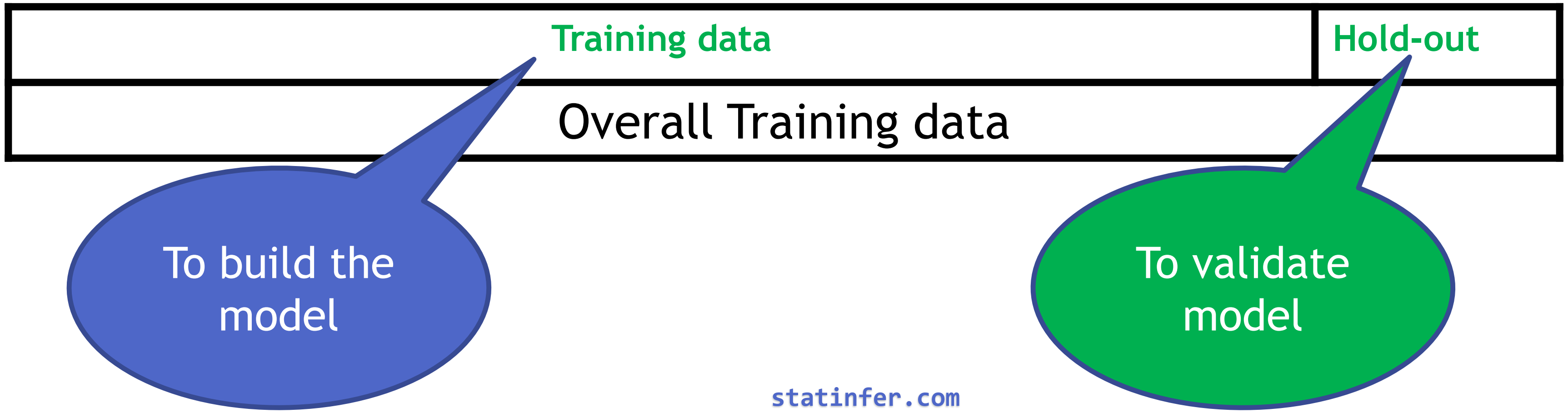
Choosing optimal model

- Choosing optimal model reduces both bias and variance.
- Unfortunately There is no standard scientific method
- How to choose optimal model complexity that gives minimum test error?
- Training error is not a good estimate of the test error.
- We can use
  - Hold out data validation
  - K-fold Cross validation methods
  - Boot strap cross validation to choose the optimal and consistent model

Holdout data Cross validation

Holdout data Cross validation

- The best solution is out of time validation. Or the testing error should be given high priority over the training error.
- A model that is performing good on training data and equally good on testing is preferred.
- We may not have to test data always. How do we estimate test error?
- We take the part of the data as training and keep aside some potion for validation. May be 80%-20% or 90%-10%
- Data splitting is a very basic intuitive method



LAB: Holdout data Cross validation

- Data: Fiberbits/Fiberbits.csv
- Take a random sample with 80% data as training sample
- Use rest 20% as holdout sample.
- Build a model on 80% of the data. Try to validate it on holdout sample.
- Try to increase or reduce the complexity and choose the best model that performs well on training data as well as holdout data

LAB: Holdout data Cross validation

```
> ###Data Splitting
> #Caret is a good package for cross validation
> library(caret)
> sampleseed <- createDataPartition(Fiberbits$active_cust, p=0.80, list=FALSE)
> train_new <- Fiberbits[sampleseed,]
> hold_out <- Fiberbits[-sampleseed,]
```

LAB: Holdout data Cross validation

```
> ########Model Building V1
> library(rpart)
> Fiber_bits_tree5<-rpart(active_cust~., method="class", control=rpart.control(minsplit=5, cp=0.
000001), data=train_new)
> Fbits_pred5<-predict(Fiber_bits_tree5, type="class")
>
> conf_matrix5<-table(Fbits_pred5,train_new$active_cust)
> conf_matrix5

Fbits_pred5     0     1
          0 31412  1718
          1  2211 44659
>
> accuracy5<-(conf_matrix5[1,1]+conf_matrix5[2,2])/(sum(conf_matrix5))
> accuracy5
[1] 0.9508875
>
> ###Validation accuracy
> hold_out$pred <- predict(Fiber_bits_tree5, hold_out, type="class")
> conf_matrix_val<-table(hold_out$pred,hold_out$active_cust)
> conf_matrix_val

        0     1
  0  7080  1391
  1  1438 10091
> accuracy_val<-(conf_matrix_val[1,1]+conf_matrix_val[2,2])/(sum(conf_matrix_val))
> accuracy_val
[1] 0.85855
```

LAB: Holdout data Cross validation

```
> ########Model Building V2
> library(rpart)
> Fiber_bits_tree5<-rpart(active_cust~., method="class", control=rpart.control(minsplit=30, c
p=0.05), data=train_new)
> Fbits_pred5<-predict(Fiber_bits_tree5, type="class")
> conf_matrix5<-table(Fbits_pred5,train_new$active_cust)
> conf_matrix5

Fbits_pred5     0     1
          0 22145  5378
          1 11478 40999
> accuracy5<-(conf_matrix5[1,1]+conf_matrix5[2,2])/(sum(conf_matrix5))
> accuracy5
[1] 0.7893
>
>
> ###Validation accuracy
> hold_out$pred <- predict(Fiber_bits_tree5, hold_out,type="class")
> conf_matrix_val<-table(hold_out$pred,hold_out$active_cust)
> conf_matrix_val

        0     1
  0  5645  1367
  1  2873 10115
> accuracy_val<-(conf_matrix_val[1,1]+conf_matrix_val[2,2])/(sum(conf_matrix_val))
> accuracy_val
[1] 0.788
```

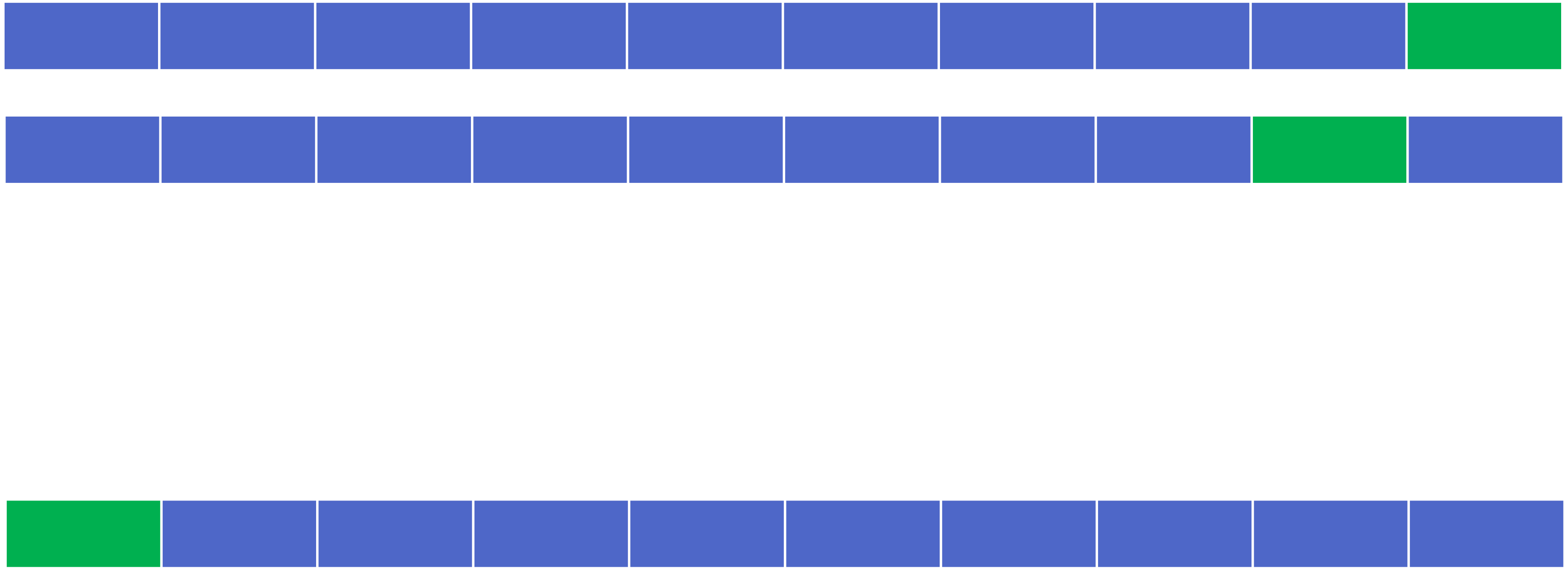
LAB: Holdout data Cross validation

```
> ########Model Building V3
> library(rpart)
> Fiber_bits_tree5<-rpart(active_cust~., method="class", control=rpart.control(minsplit=30, c
p=0.01),   data=train_new)
>
> library(rattle)
> fancyRpartPlot(Fiber_bits_tree5)
>
> Fbits_pred5<-predict(Fiber_bits_tree5, type="class")
> conf_matrix5<-table(Fbits_pred5,train_new$active_cust)
> conf_matrix5

Fbits_pred5     0     1
          0 25099  3825
          1  8524 42552
> accuracy5<-(conf_matrix5[1,1]+conf_matrix5[2,2])/(sum(conf_matrix5))
> accuracy5
[1] 0.8456375
>
> ###Validation accuracy
> hold_out$pred <- predict(Fiber_bits_tree5, newdata=hold_out,type="class")
> conf_matrix_val<-table(hold_out$pred,hold_out$active_cust)
> conf_matrix_val

        0     1
  0  6437   941
  1  2081 10541
> accuracy_val<-(conf_matrix_val[1,1]+conf_matrix_val[2,2])/(sum(conf_matrix_val))
> accuracy_val
[1] 0.8489
```

Ten-fold Cross - Validation

Ten-fold Cross - Validation

- Divide the data into 10 parts(randomly)
- Use 9 parts as training data(90%) and the tenth part as holdout data(10%)
- We can repeat this process 10 times
- Build 10 models, find average error on 10 holdout samples. This gives us an idea on testing error



statinfer

K-fold - Validation

K-fold Cross Validation

- A generalization of cross validation.
- Divide the whole dataset into k equal parts
- Use k^{th} part of the data as the holdout sample, use remaining k-1 parts of the data as training data
- Repeat this K times, build K models. The average error on holdout sample gives us an idea on the testing error
- Which model to choose?
  - Choose the model with least error and least complexity
  - Or the model with less than average error and simple (less parameters)
  - Finally use complete data and build a model with the chosen number of parameters
- Note: Its better to choose K between 5 to 10. Which gives 80% to 90% training data and rest 20% to 10% is holdout data

statinfer

LAB- K-fold Cross Validation

LAB- K-fold Cross Validation

- Build a tree model on the fiber bits data.
- Try to build the best model by making all the possible adjustments to the parameters.
- What is the accuracy of the above model?
- Perform 10 –fold cross validation. What is the final accuracy?
- Perform 20 –fold cross validation. What is the final accuracy?
- What can be the expected accuracy on the unknown dataset?

LAB- K-fold Cross Validation

```
> ##################################
> #k-fold Cross Validation
>
> library(rpart)
>
> #########Overfitting
> ###Model on complete training data
> Fiber_bits_tree3<-rpart(active_cust~., method="class", control=rpart.control(minsplit=10, c
p=0.000001), data=Fiberbits)
> Fbits_pred3<-predict(Fiber_bits_tree3, type="class")
> conf_matrix3<-table(Fbits_pred3,Fiberbits$active_cust)
> conf_matrix3

Fbits_pred3     0     1
          0 38154  2849
          1  3987 55010
> accuracy3<-(conf_matrix3[1,1]+conf_matrix3[2,2])/(sum(conf_matrix3))
> accuracy3
[1] 0.93164
```

LAB- K-fold Cross Validation

```
> #k-fold Cross Validation building
> #######K=10
> library(caret)
> train_dat <- trainControl(method="cv", number=10)
>
> #Need to convert the dependent variable to factor before fitting the model
> Fiberbits$active_cust<-as.factor(Fiberbits$active_cust)
>
> #Building the models on K-fold  samples
> K_fold_tree<-train(active_cust~., method="rpart", trControl=train_dat, control=rpart.contro
l(minsplit=10, cp=0.000001),  data=Fiberbits)
> K_fold_tree
CART

1e+05 samples
8e+00 predictors
2e+00 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 90000, 90001, 90000, 90000, 90000, 90000, ...
Resampling results across tuning parameters:

  cp         Accuracy   Kappa
  0.0876581  0.7735096  0.5276540
  0.1639971  0.7095007  0.3616703
  0.2477397  0.6504500  0.1936002

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was cp = 0.0876581.
```

LAB- K-fold Cross Validation

```
> K_fold_tree$finalModel
n= 100000

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 100000 42141 1 (0.42141000 0.57859000)
  2) relocated>=0.5 12348   954 0 (0.92274052 0.07725948) *
  3) relocated< 0.5 87652 30747 1 (0.35078492 0.64921508)
    6) Speed_test_result< 78.5 27517 10303 0 (0.62557692 0.37442308) *
    7) Speed_test_result>=78.5 60135 13533 1 (0.22504365 0.77495635) *
>
> library(rattle)
> fancyRpartPlot(K_fold_tree$finalModel)
>
> Kfold_pred<-predict(K_fold_tree)
> #Caret package has confusion matrix function
> conf_matrix6<-confusionMatrix(Kfold_pred,Fiberbits$active_cust)
```

LAB- K-fold Cross Validation

```
> conf_matrix6
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 28608 11257
         1 13533 46602

               Accuracy : 0.7521
                 95% CI : (0.7494, 0.7548)
    No Information Rate : 0.5786
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4879
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6789
            Specificity : 0.8054
         Pos Pred Value : 0.7176
         Neg Pred Value : 0.7750
             Prevalence : 0.4214
         Detection Rate : 0.2861
   Detection Prevalence : 0.3987
      Balanced Accuracy : 0.7422

       'Positive' Class : 0
```

LAB- K-fold Cross Validation

```
> #######K=20
> library(caret)
> train_dat <- trainControl(method="cv", number=20)
>
> #Need to convert the dependent variable to factor before fitting the model
> Fiberbits$active_cust<-as.factor(Fiberbits$active_cust)
>
> #Building the models on K-fold  samples
> K_fold_tree_1<-train(active_cust~., method="rpart", trControl=train_dat, control=rpart.cont
rol(minsplit=10, cp=0.000001),  data=Fiberbits)
> K_fold_tree_1$finalModel
n= 100000

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 100000 42141 1 (0.42141000 0.57859000)
  2) relocated>=0.5 12348   954 0 (0.92274052 0.07725948) *
  3) relocated< 0.5 87652 30747 1 (0.35078492 0.64921508)
    6) Speed_test_result< 78.5 27517 10303 0 (0.62557692 0.37442308) *
    7) Speed_test_result>=78.5 60135 13533 1 (0.22504365 0.77495635) *
>
> library(rattle)
> fancyRpartPlot(K_fold_tree_1$finalModel)
>
> Kfold_pred<-predict(K_fold_tree_1)
> #Caret package has confusion matrix function
> conf_matrix6_1<-confusionMatrix(Kfold_pred,Fiberbits$active_cust)
```

LAB- K-fold Cross Validation

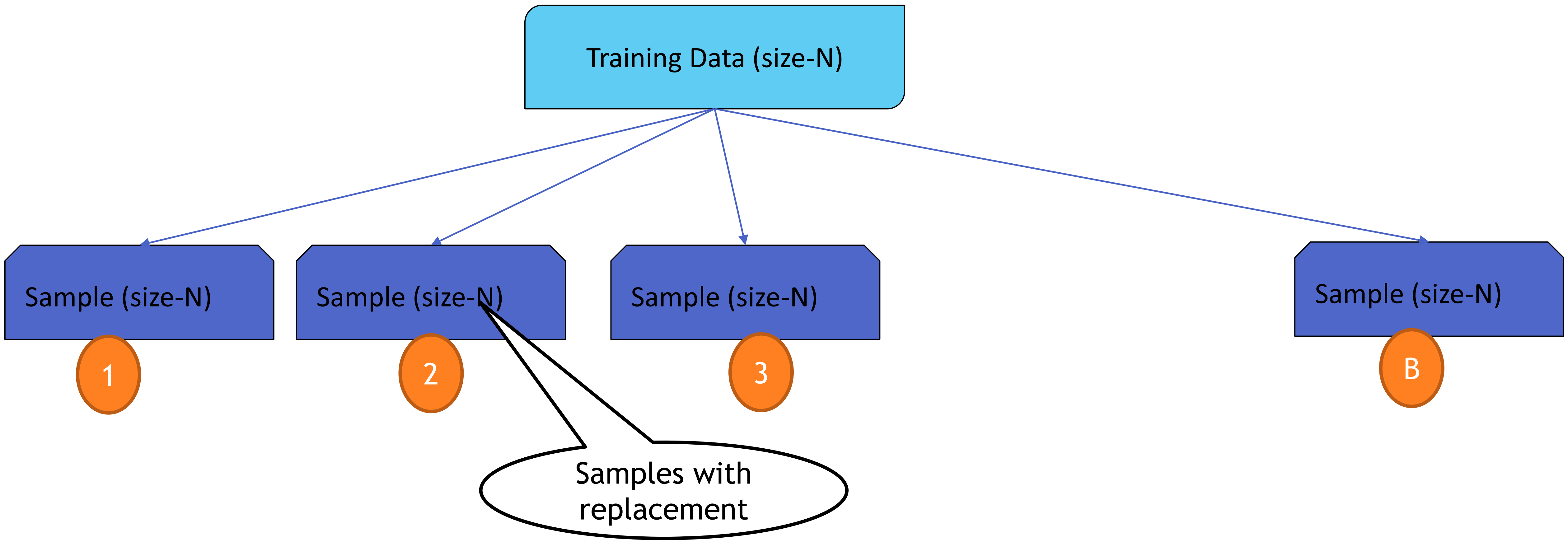
```
> conf_matrix6_1
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 28608 11257
         1 13533 46602

               Accuracy : 0.7521
                 95% CI : (0.7494, 0.7548)
    No Information Rate : 0.5786
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4879
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6789
            Specificity : 0.8054
         Pos Pred Value : 0.7176
         Neg Pred Value : 0.7750
             Prevalence : 0.4214
         Detection Rate : 0.2861
   Detection Prevalence : 0.3987
      Balanced Accuracy : 0.7422

       'Positive' Class : 0
```

Bootstrap Cross Validation

Bootstrap Methods

- What if we have just 100 observations overall. If we do a 10 fold cross validation then each part has only 10 observations.
- K-Fold might fail while validating the models in each iteration.
- Boot strapping is a powerful tool to get an idea on accuracy of the model and the test error, especially when dataset size is small.
- Can estimate the likely future performance of a given modeling procedure, on new data not yet realized.

Bootstrap Method



The Algorithm

- We have a training data is of size N
- Draw random sample with replacement of size N – This gives a new dataset, it might have repeated observations, some observations might not have even appeared once.
- Create B such new datasets. These are called boot strap datasets
- Build the model on these B datasets, we can test the models on the original training dataset.

Bootstrap Example

- Example
  1. We have a training data is of size 500
  2. Boot Strap Data-1: Create a dataset of size 500. To create this dataset, draw **a random point**, note it down, then replace it back. Again draw another sample point. Repeat this process 500 times. This makes a dataset of size 500. Call this as Boot Strap Data-1
  3. Multiple Boot Strap datasets :Repeat the procedure in step -2 multiple times. Say 200 times. Then we have 200 Boot Strap datasets
  4. We can build the models on these 200 boost strap datasets and the average error gives a good idea on overall error.
  5. We can even use the original training data as the test data for each of the models

LAB: Bootstrap cross validation

- Create a new dataset by taking a random sample of size 250; Name it as fiber_sample_data
- In fiber_sample_data, draw a boot strap sample with sufficient sample size
- Build a tree model and get an estimate on true accuracy of the model

Code: Bootstrap cross validation

```
> #######Bootstrap
> library(caret)
> train_control <- trainControl(method="boot", number=10)
> #Where number is the number of bootstrap sample sets i.e B
> ###Tree model on boots straped data
> Boot_Strap_model <- train(active_cust~., method="rpart", trControl=train_dat, control=rpart
.control(minsplit=10, cp=0.000001),  data=Fiberbits)
> Boot_Strap_model$finalModel
n= 100000

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 100000 42141 1 (0.42141000 0.57859000)
  2) relocated>=0.5 12348   954 0 (0.92274052 0.07725948) *
  3) relocated< 0.5 87652 30747 1 (0.35078492 0.64921508)
    6) Speed_test_result< 78.5 27517 10303 0 (0.62557692 0.37442308) *
    7) Speed_test_result>=78.5 60135 13533 1 (0.22504365 0.77495635) *
```

Code: Bootstrap cross validation

```
> Boot_Strap_predictions <- predict(Boot_Strap_model)
> conf_matrix7<-confusionMatrix(Boot_Strap_predictions,Fiberbits$active_cust)
> conf_matrix7
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 28608 11257
         1 13533 46602

               Accuracy : 0.7521
                 95% CI : (0.7494, 0.7548)
    No Information Rate : 0.5786
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4879
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6789
            Specificity : 0.8054
         Pos Pred Value : 0.7176
         Neg Pred Value : 0.7750
             Prevalence : 0.4214
         Detection Rate : 0.2861
   Detection Prevalence : 0.3987
      Balanced Accuracy : 0.7422

       'Positive' Class : 0
```

Other Validation Metrics

F1 – Score

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

	Predicted Classes: 0(Positive)	Predicted Classes: 1(Negative)
Actual Classes 0(Positive)	True positive (TP) Actual condition is Positive, it is truly predicted as positive	False Negatives(FN) Actual condition is Positive, it is falsely predicted as negative
1(Negative)	False Positives(FP) Actual condition is Negative, it is falsely predicted as positive	True Negatives(TN) Actual condition is Negative, it is truly predicted as negative

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

- Recall is a fraction and Precision is a fraction.
- F1 score is Harmonic mean of Recall and Precision

When to use F1 – Score

- F1 score need to be calculated separately for individual classes
- Use it while dealing with while dealing with imbalanced classes. Where one class really dominates the other
- Use F1 score in It is very useful for multi class problems. F1 is also known as per class accuracy.

- Limitations:
  - Different values of F1 score for different threshold values
  - It is very difficult to set threshold for F1 score.

LAB: F1-Score

- Product Sales Data/Product_sales.csv
- Build a logistic regression model on product sales data.
- Create a confusing matrix.
- Calculate F1 score

Code: F1-Score

```
> prod_sales_Logit_model <- glm(Bought ~ Age,family=binomial(),data=Product_sales)
>
> #Confusion matrix
> threshold=0.5
> predicted_class<-ifelse(predict(prod_sales_Logit_model,type="response")>threshold,1,0)
>
> actual_values<-prod_sales_Logit_model$y
> conf_matrix<-table(actual_values,predicted_class)
> conf_matrix
             predicted_class
actual_values   0   1
            0 257   5
            1   3 202
>
> #F1 Score
> library(MLmetrics)
>
> #F1 score of "0" class
> F1_Score(actual_values, predicted_class, positive = 0)
[1] 0.9846743
>
> #F1 score of "1" class
> F1_Score(actual_values, predicted_class, positive = 1)
[1] 0.9805825
```

Many More

1. FBeta_Score
2. GainAUC
3. Gini
4. KS_Stat
5. Accuracy LiftAUC
6. LogLoss
7. MAE
8. MAPE
9. MSE
10. MultiLogLoss
11. NormalizedGini
12. Poisson_LogLoss
13. PRAUC
14. R2_Score
15. RMSE
16. ZeroOneLoss
17. Kappa

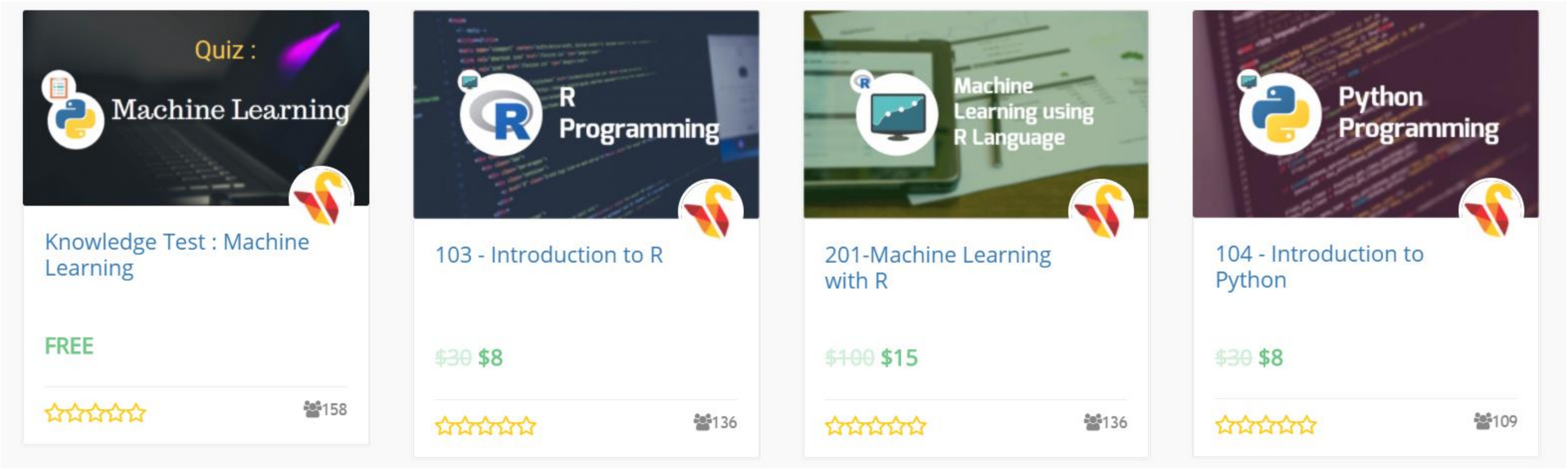
Conclusion

Conclusion

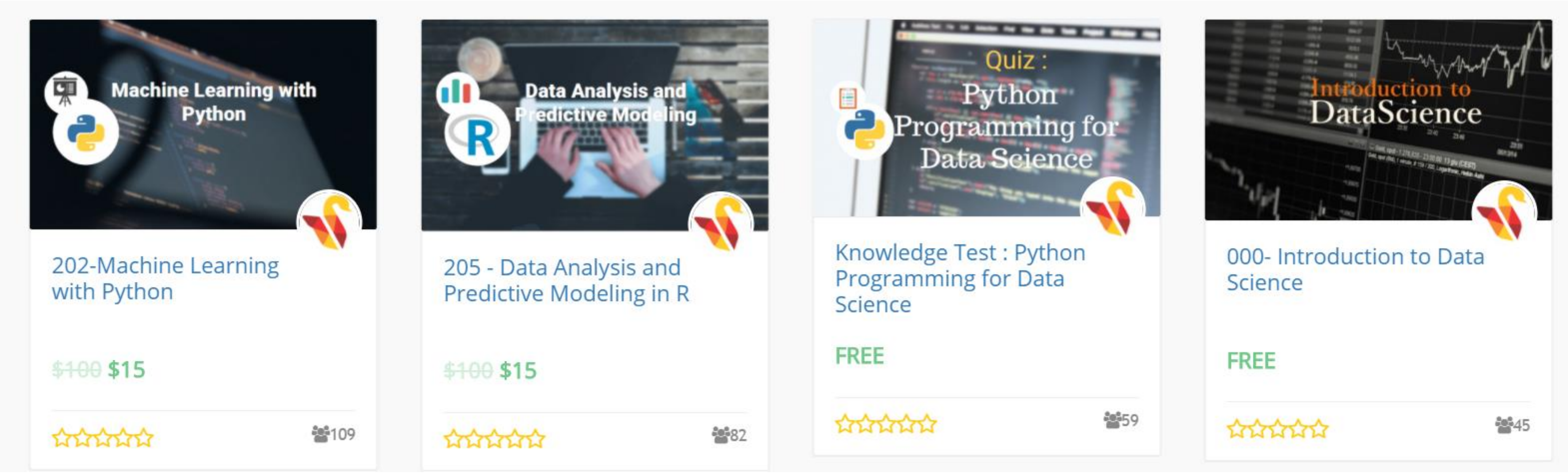
- We studied
  - Validating a model, Types of data & Types of errors
  - The problem of over fitting & The problem of under fitting
  - Bias Variance Tradeoff
  - Cross validation & Boot strapping
- Training error is what we see and that is not the true performance metric
- Test error plays vital role in model selection
- Cross Validation and Boot strapping techniques give us an idea on test error
- Choose the model based on the combination of Accuracy, AIC, Cross Validation and Boot strapping results
- Bootstrap is widely used in ensemble models & random forests.

Thank you

All our courses are available at nominal prices



All our courses are available at nominal prices



Statinfer.com

Data Science Training and R&D

Training on

Data Science

Bigdata Analytics

Machine Learning

Predictive Modelling

Deep Learning

Corporate Training

Classroom Training

Online Training

Contact us

info@statinfer.com

venkat@statinfer.com