Model Selection and Cross Validation

Contents

- How to validate a model?
- What is a best model ?
- Types of data
- Types of errors
- The problem of over fitting
- The problem of under fitting
- Bias Variance Tradeoff
- Cross validation
- Boot strapping

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
- KS, Gini
- Concordance and discordance
- Chi-Square, Hosmer and Lemeshow Goodness-of-Fit Test
- Lift curve

- 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

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		
		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	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

Calculating Sensitivity and Specificity

LAB- Sensitivity and Specificity

1. Build a logistic regression model on fiber bits data
2. Create the confusion matrix
3. Find the accuracy
4. Calculate Specificity
5. Calculate Sensitivity

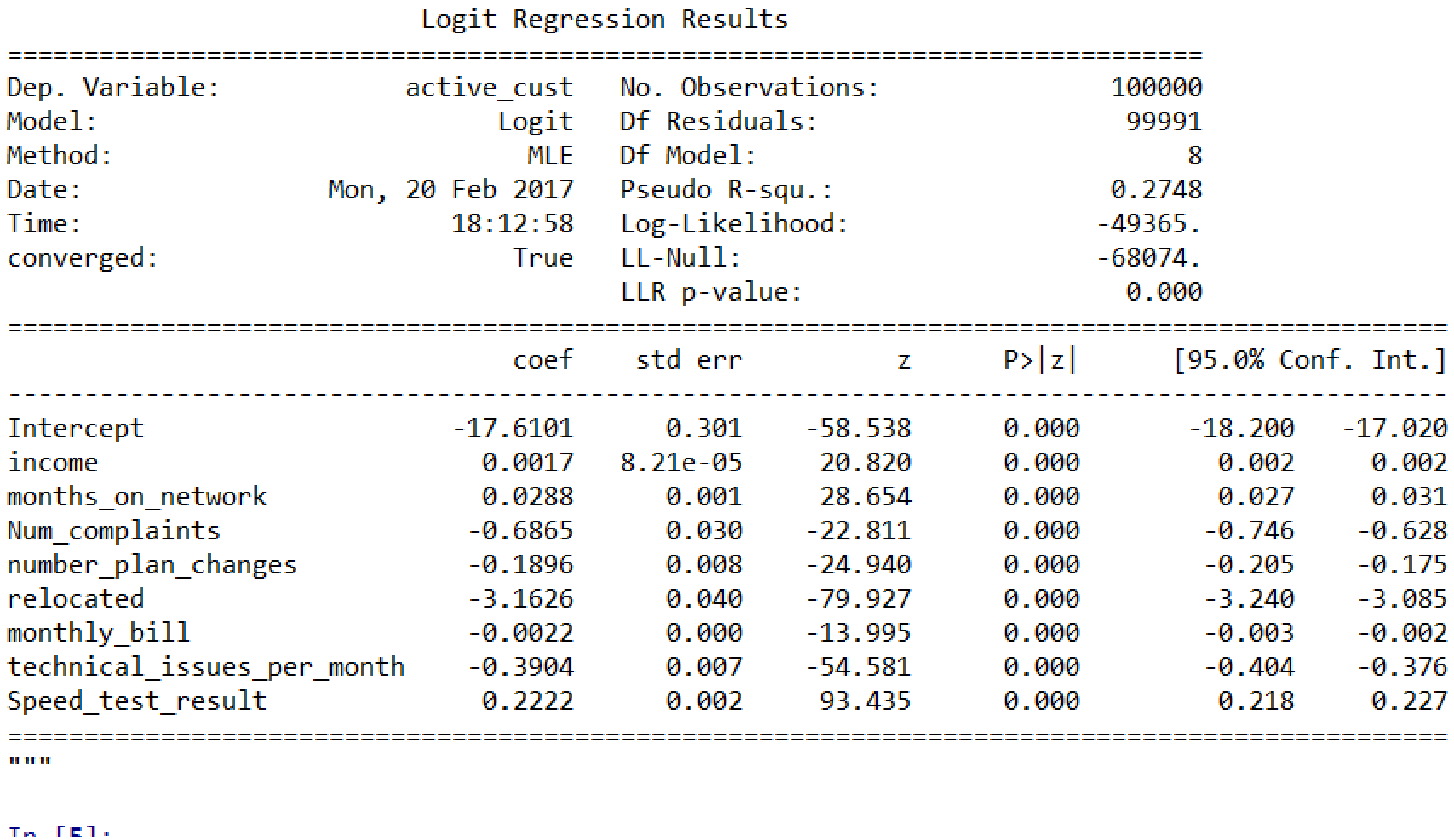
Code Sensitivity and Specificity

1) Build a logistic regression model on fiber bits data

```
import sklearn as sk
import pandas as pd
import numpy as np
import scipy as sp

Fiber_df= pd.read_csv("D:\\Google Drive\\Training\\Datasets\\Fiberbits\\Fiberbits.csv")
Fiber_df.head(5)
Fiber_df.tail(5)

from sklearn.linear_model import LogisticRegression
import statsmodels.formula.api as sm

#logistic1= LogisticRegression()
logistic1 = sm.logit(formula='active_cust~income+months_on_network+Num_complaints+number_plan_changes+relocated+monthly_bill+technical_issues_per_month+Speed_test_result', data=Fiber_df)
fitted1 =logistic1.fit()
fitted1.summary()
```

Code Sensitivity and Specificity

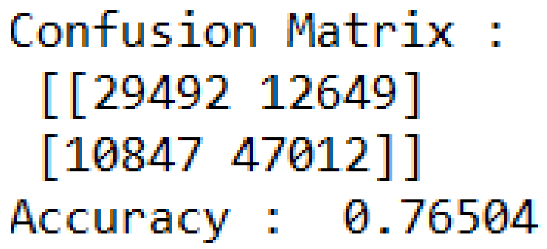
1) Build a logistic regression model on fiber bits data-



```
                           Logit Regression Results
==============================================================================
Dep. Variable:            active_cust   No. Observations:               100000
Model:                          Logit   Df Residuals:                    99991
Method:                           MLE   Df Model:                            8
Date:                Mon, 20 Feb 2017   Pseudo R-squ.:                  0.2748
Time:                        18:12:58   Log-Likelihood:                -49365.
converged:                       True   LL-Null:                       -68074.
                                        LLR p-value:                     0.000
==============================================================================================
                                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
----------------------------------------------------------------------------------------------
Intercept                    -17.6101      0.301    -58.538      0.000       -18.200   -17.020
income                         0.0017   8.21e-05     20.820      0.000         0.002     0.002
months_on_network              0.0288      0.001     28.654      0.000         0.027     0.031
Num_complaints                -0.6865      0.030    -22.811      0.000        -0.746    -0.628
number_plan_changes           -0.1896      0.008    -24.940      0.000        -0.205    -0.175
relocated                     -3.1626      0.040    -79.927      0.000        -3.240    -3.085
monthly_bill                  -0.0022      0.000    -13.995      0.000        -0.003    -0.002
technical_issues_per_month    -0.3904      0.007    -54.581      0.000        -0.404    -0.376
Speed_test_result              0.2222      0.002     93.435      0.000         0.218     0.227
==============================================================================================
"""


In [5]:
```

Code Sensitivity and Specificity

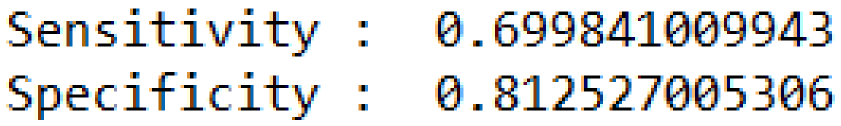
```
#####Create the confusion matrix
###predict the variable active customer from logistic fit####
predicted_values1=fitted1.predict(Fiber_df[["income"]+['months_on_network']+['Num_complaints']+['number_plan_changes']+['relocated']+['monthly_bill']+['technical_issues_per_month']+['Speed_test_result']])
predicted_values1[1:10]

### Converting predicted values into classes using threshold
threshold=0.5
predicted_class1=np.zeros(predicted_values1.shape)
predicted_class1[predicted_values1>threshold]=1
predicted_class1

from sklearn.metrics import confusion_matrix
cm1= confusion_matrix(Fiber_df[['active_cust']],predicted_class1)
print('Confusion Matrix : \n', cm1)

total1=sum(sum(cm1))
#####from confusion matrix calculate accuracy
accuracy1=(cm1[0,0]+cm1[1,1])/total1
print ('Accuracy : ', accuracy1)
```



```
Confusion Matrix :
 [[29492 12649]
 [10847 47012]]
Accuracy :  0.76504
```

Code Sensitivity and Specificity

```
sensitivity1 =cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ', sensitivity1 )

specificity1 =cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ', specificity1)
```



```
Sensitivity :  0.699841009943
Specificity :  0.812527005306
```

Sensitivity vs Specificity

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

		Predicted Classes		
		0(Yes-Defaulter)	1(Non-Defaulter)	
Actual Classes	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

		0(Yes-Defaulter)	1(Non-Defaulter)	
Actual Classes	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

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

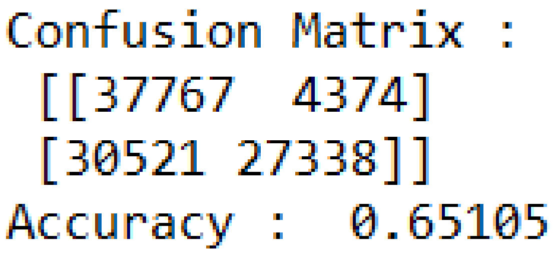
- There are some cases where Sensitivity is important and need to be near to 1
- There are business cases where Specificity is important and need to be near to 1
- We need to understand the business problem and decide the importance of Sensitivity and Specificity

LAB: Sensitivity vs Specificity with Different Thresholds

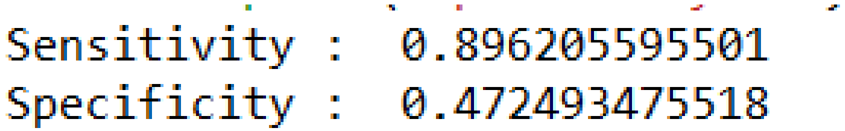
- Try different thresholds and see the change in sensitivity and specificity
  - Try Low threshold value
  - Try high threshold value

Code: Sensitivity vs Specificitywith Different Thresholds

```
threshold=0.8

predicted_class1=np.zeros(predicted_values1.shape)
predicted_class1[predicted_values1>threshold]=1
predicted_class1

#Change in Confusion Matrix, Accuracy and Sensitivity-Specificity
#Confusion matrix, Accuracy, sensitivity and specificity
from sklearn.metrics import confusion_matrix

cm1=confusion_matrix(Fiber_df[['active_cust']],predicted_class1)
print('Confusion Matrix : \n', cm1)

total1=sum(sum(cm1))
#####from confusion matrix calculate accuracy
accuracy1=(cm1[0,0]+cm1[1,1])/total1
print ('Accuracy : ', accuracy1)

sensitivity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ', sensitivity1 )

specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ', specificity1)
```



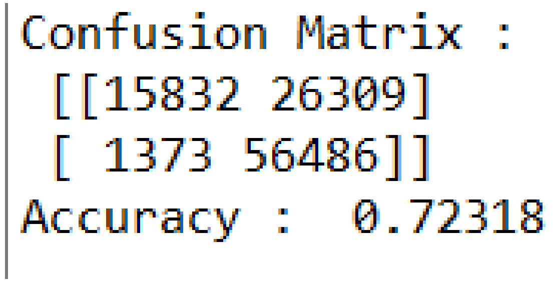
```
Confusion Matrix :
 [[37767  4374]
 [30521 27338]]
Accuracy :  0.65105
```



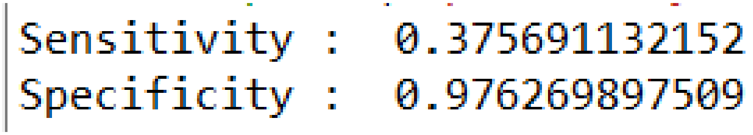
```
Sensitivity :  0.896205595501
Specificity :  0.472493475518
```

Code: Sensitivity vs Specificitywith Different Thresholds

```
###A low threshold value
threshold=0.3

predicted_class1=np.zeros(predicted_values1.shape)
predicted_class1[predicted_values1>threshold]=1
predicted_class1

#Change in Confusion Matrix, Accuracy and Sensitivity-Specificity
#Confusion matrix, Accuracy, sensitivity and specificity
from sklearn.metrics import confusion_matrix

cm1=confusion_matrix(Fiber_df[['active_cust']],predicted_class1)
print('Confusion Matrix : \n', cm1)

total1=sum(sum(cm1))
#####from confusion matrix calculate accuracy
accuracy1=(cm1[0,0]+cm1[1,1])/total1
print ('Accuracy : ', accuracy1)

sensitivity1 =cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ', sensitivity1 )

specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ', specificity1)
```



```
Confusion Matrix :
 [[15832 26309]
 [ 1373 56486]]
Accuracy :  0.72318
```

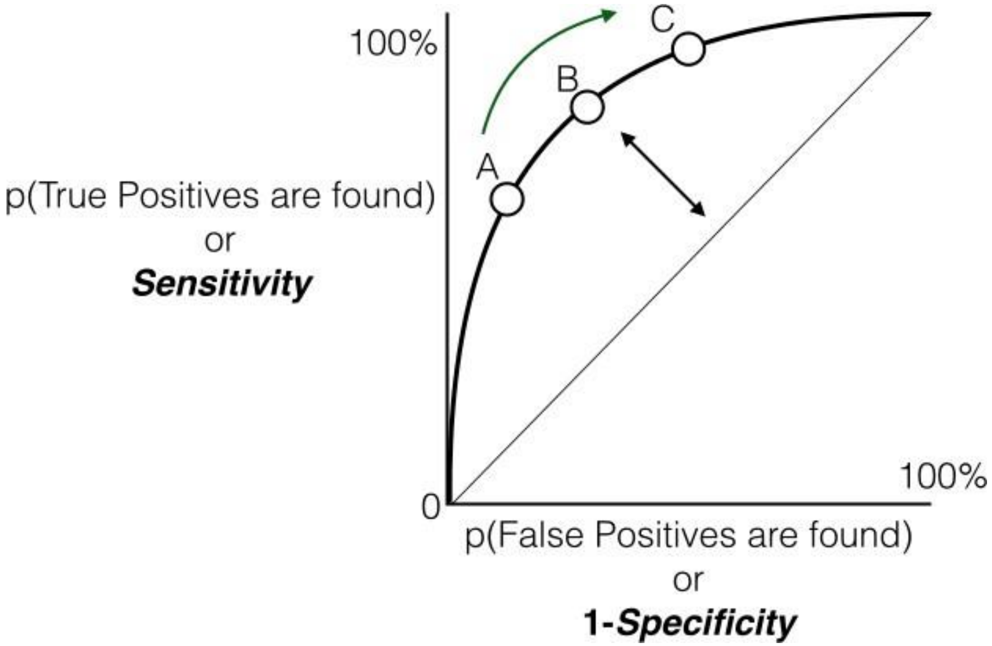


```
Sensitivity :  0.375691132152
Specificity :  0.976269897509
```

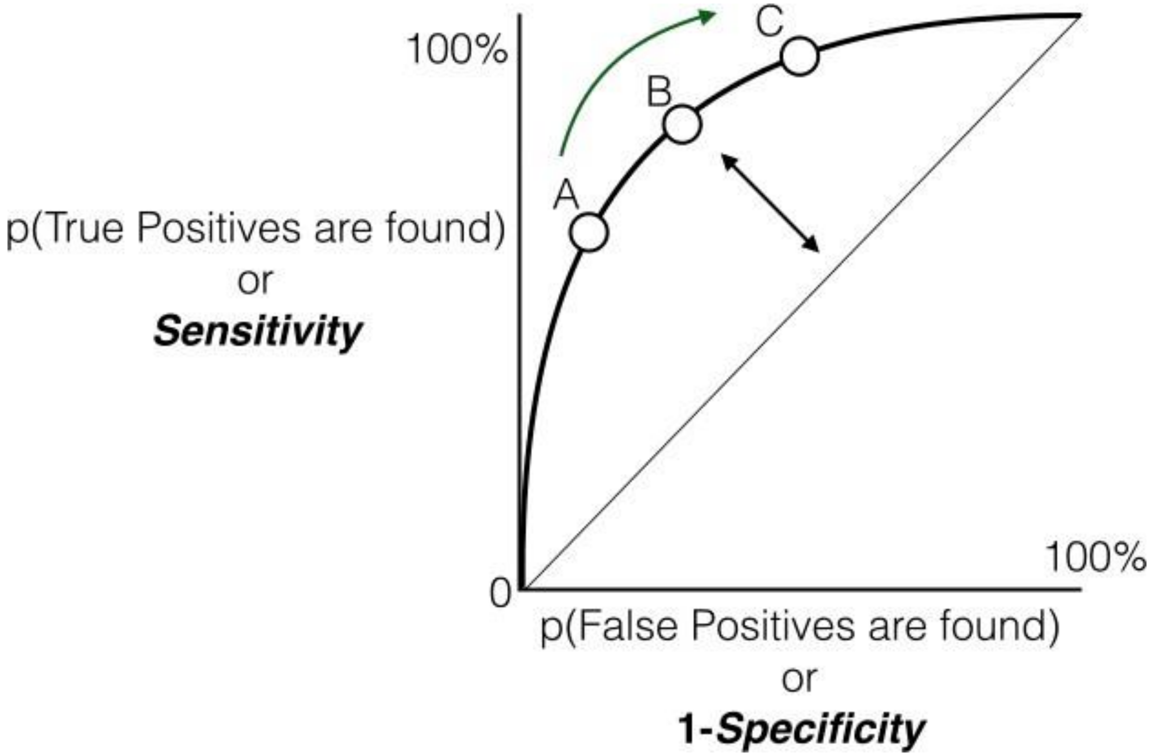
ROC Curve

ROC Curve

- If we consider all the possible threshold values and the corresponding specificity and sensitivity rate what will be the final model accuracy.
- ROC(Receiver operating characteristic) curve is drawn by taking False positive rate on X-axis and True positive rate on Y- axis
- ROC tells us, how many mistakes are we making to identify all the positives?



ROC Curve -Interpretation

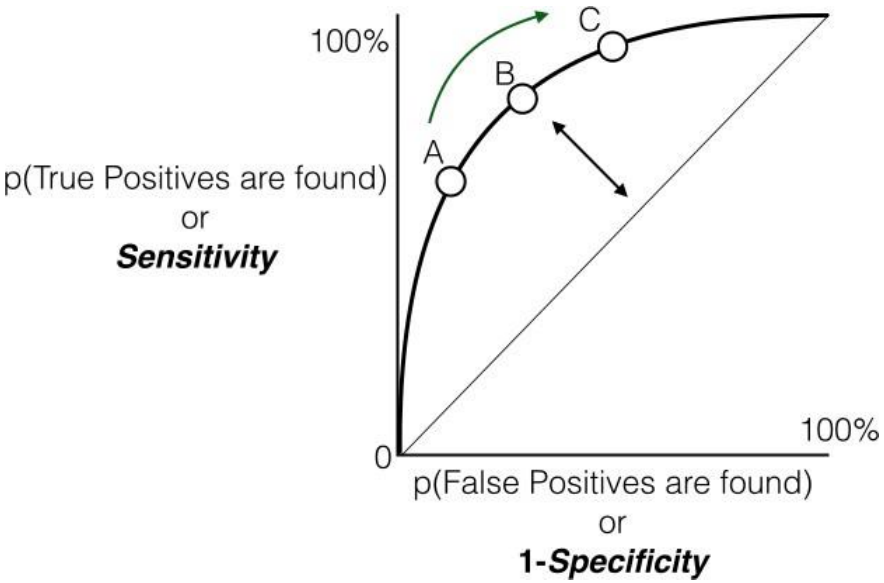


- How many mistakes are we making to identify all the positives?
- How many mistakes are we making to identify 70%, 80% and 90% of positives?
- 1-Specificty(false positive rate) gives us an idea on mistakes that we are making
- We would like to make 0% mistakes for identifying 100% positives
- We would 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

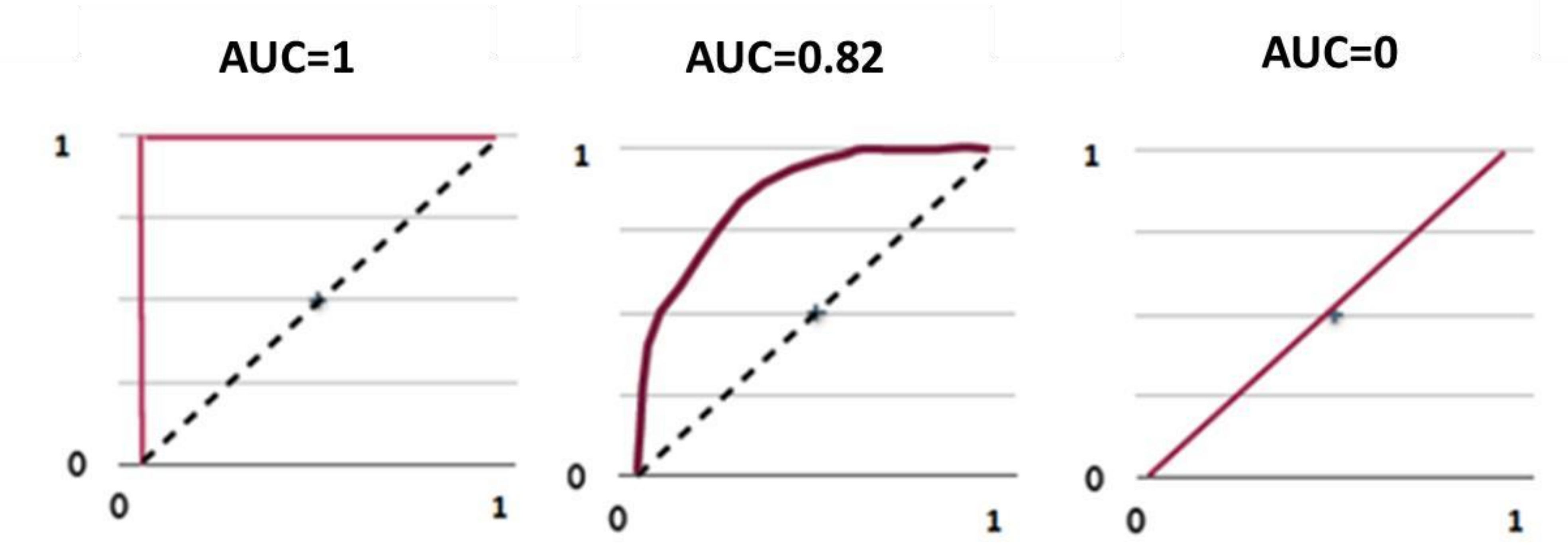
ROC and AUC

- 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



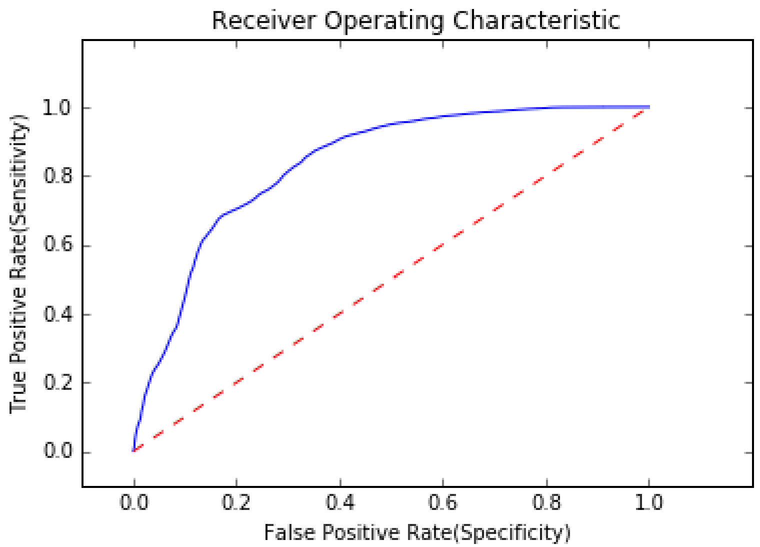
ROC and AUC Calculation

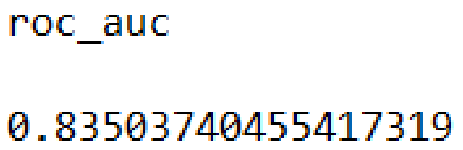
LAB: ROC and AUC

- Calculate ROC and AUC for fiber bits logistic regression model

LAB: ROC and AUC

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

actual = Fiber_df[['active_cust']]
false_positive_rate, true_positive_rate, thresholds = roc_curve(actual,
predicted_values1)
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate(Sensitivity)')
plt.xlabel('False Positive Rate(Specificity)')
plt.show()

###Area under Curve-AUC
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```





```
roc_auc

0.83503740455417319
```

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 accuratemodel

- 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

```
features = list(Fiber_df.drop(['active_cust'],1).columns) #this code gives a list
of column names except 'active_cust'

X= np.array(Fiber_df[features])
y = np.array(Fiber_df['active_cust'])

from sklearn import tree
#Let's make a model by choosing some initial parameters.
tree_config = tree.DecisionTreeClassifier(criterion='gini',
                                          splitter='best',
                                          max_depth=10,
                                          min_samples_split=1,
                                          min_samples_leaf=30,
                                          max_leaf_nodes=10)

#What is the accuracy of your model?
tree_config.fit(X,y)
tree_config.score(X,y)
```

```
Accuracy
0.84972999999999999
```

Code: The most accurate model

```
#Grow the tree as much as you can and achieve 90% accuracy.
#Let's make a model by changing the parameters to increases accuracy

tree_config_new = tree.DecisionTreeClassifier(criterion='gini',
                                              splitter='best',
                                              max_depth=None,
                                              min_samples_split=2,
                                              min_samples_leaf=1,
                                              max_leaf_nodes=None)
tree_config_new.fit(X,y)
tree_config_new.score(X,y)
```

```
Accuracy
0.99668999999999996
```

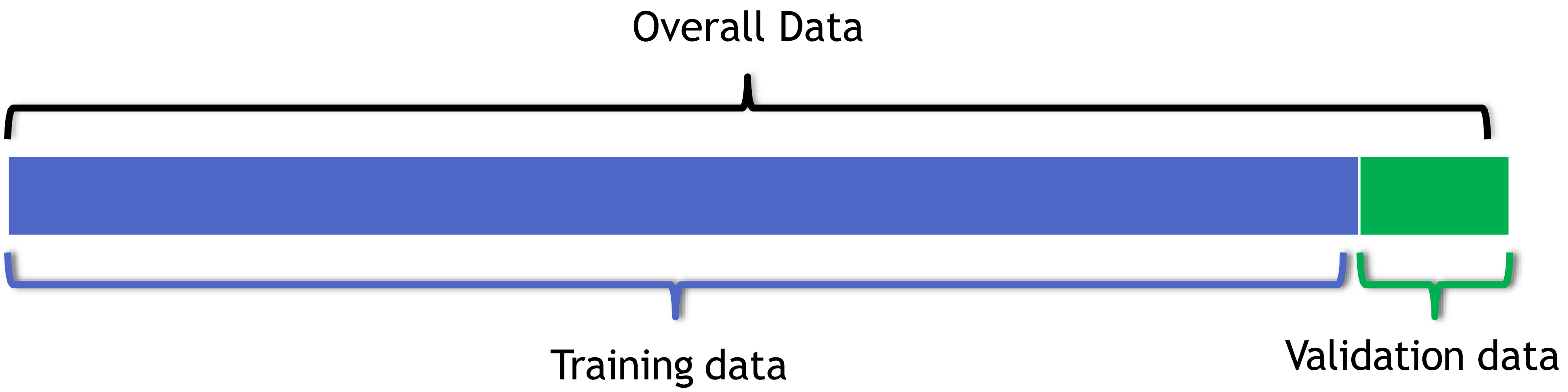
Different type of datasets and errors

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

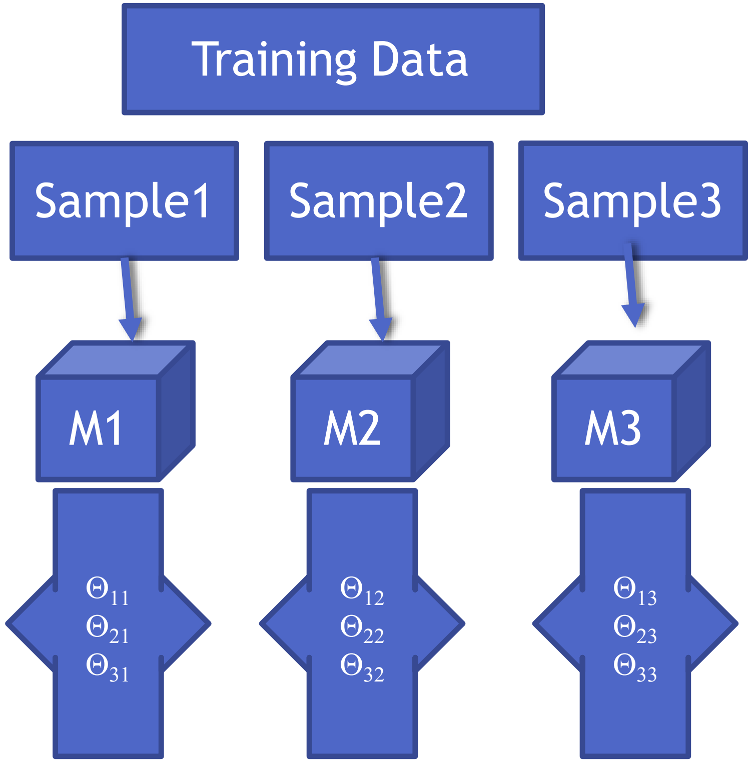
- The training error
  - The error on training dataset
  - In-time error
  - Error on the known data
  - Can be reduced while building the model
- The test error
  - The error that matters
  - Out-of-time error
  - The error on unknown/new dataset.

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

The problem of overfitting

The problem of overfitting

- 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.
- Imagine building multiple models with small changes in training data. The resultant set of models will have huge variance in their parameter estimates.



Training Data

Sample1 Sample2 Sample3

M1 M2 M3

Θ_{11} Θ_{21} Θ_{31}

Θ_{12} Θ_{22} Θ_{32}

Θ_{13} Θ_{23} Θ_{33}

The problem of overfitting

- The model is made really complicated, that it is very sensitive to minimal changes
- 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?

Code: Model with huge Variance

```
X= np.array(Fiber_df[features])
y = np.array(Fiber_df['active_cust'])

from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test =train_test_split(X,y, train_size =0.9)
#Build the best model(5% error) model on training data.
tree_var = tree.DecisionTreeClassifier(criterion='gini',
                                        splitter='best',
                                        max_depth=20,
                                        min_samples_split=2,
                                        min_samples_leaf=1,
                                        max_leaf_nodes=None)
tree_var.fit(X_train,y_train)

tree_config_new.fit(X_train,y_train)
tree_config_new.score(X_train,y_train)
```

```
 Accuracy
0.99711111111111106
```

Code: Model with huge Variance

```
#Use the validation data to verify the error rate. Is the error rate on the training data and validation data same?
predict_test = tree_config_new.predict(X_test)
print(predict_test)

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,predict_test)
total = sum(sum(cm))
accuracy = (cm[0,0]+cm[1,1])/total
print(accuracy)
```

```
Accuracy
0.8467
```

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 hugeBias

- 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.

Code: Model with huge Bias

```
#Lets prune the tree further.  Lets oversimplyfy the model
tree_bias1 = tree.DecisionTreeClassifier(criterion='gini',
                                         splitter='random',
                                         max_depth=1,
                                         min_samples_split=100,
                                         min_samples_leaf=100,
                                         max_leaf_nodes=2)
tree_bias1.fit(X_train,y_train)


#Training Accuracy of new model
tree_bias1.score(X_train,y_train)

#Validation Error
tree_bias1.score(X_test,y_test)
print(predict_test)

#Validation accuracy on test data
tree_bias1.score(X_test,y_test)
```

Training Accuracy
0.68271111111111116

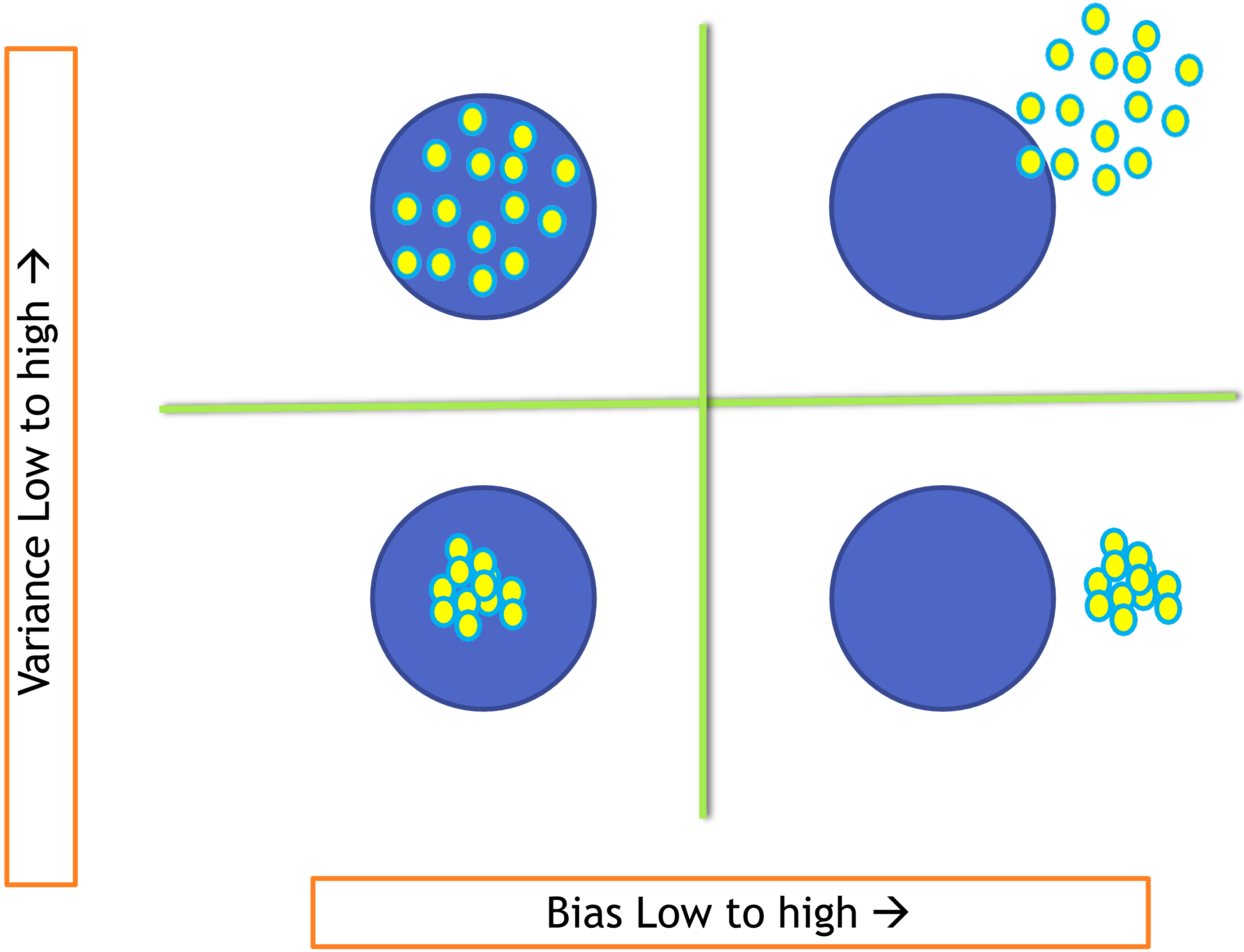
Testing Accuracy
0.6855

Model Bias and Variance

Model Bias andVariance

- 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 andVariance



Variance Low to high →

Bias Low to high →

Model aim is to hit
the center of circle

The Bias-Variance Decomposition

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

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

$$SquaredError = E[(Y - \hat{f}(x_0))_2 \mid X = x_0]$$

$$= \sigma^2 + [E\hat{f}(x_0) - f(x_0)]^2 + E[\hat{f}(x_0) - E\hat{f}(x_0)]^2$$

$$= \sigma^2 + Bias^2(\hat{f}(x_0)) + Var(\hat{f}(x_0))$$

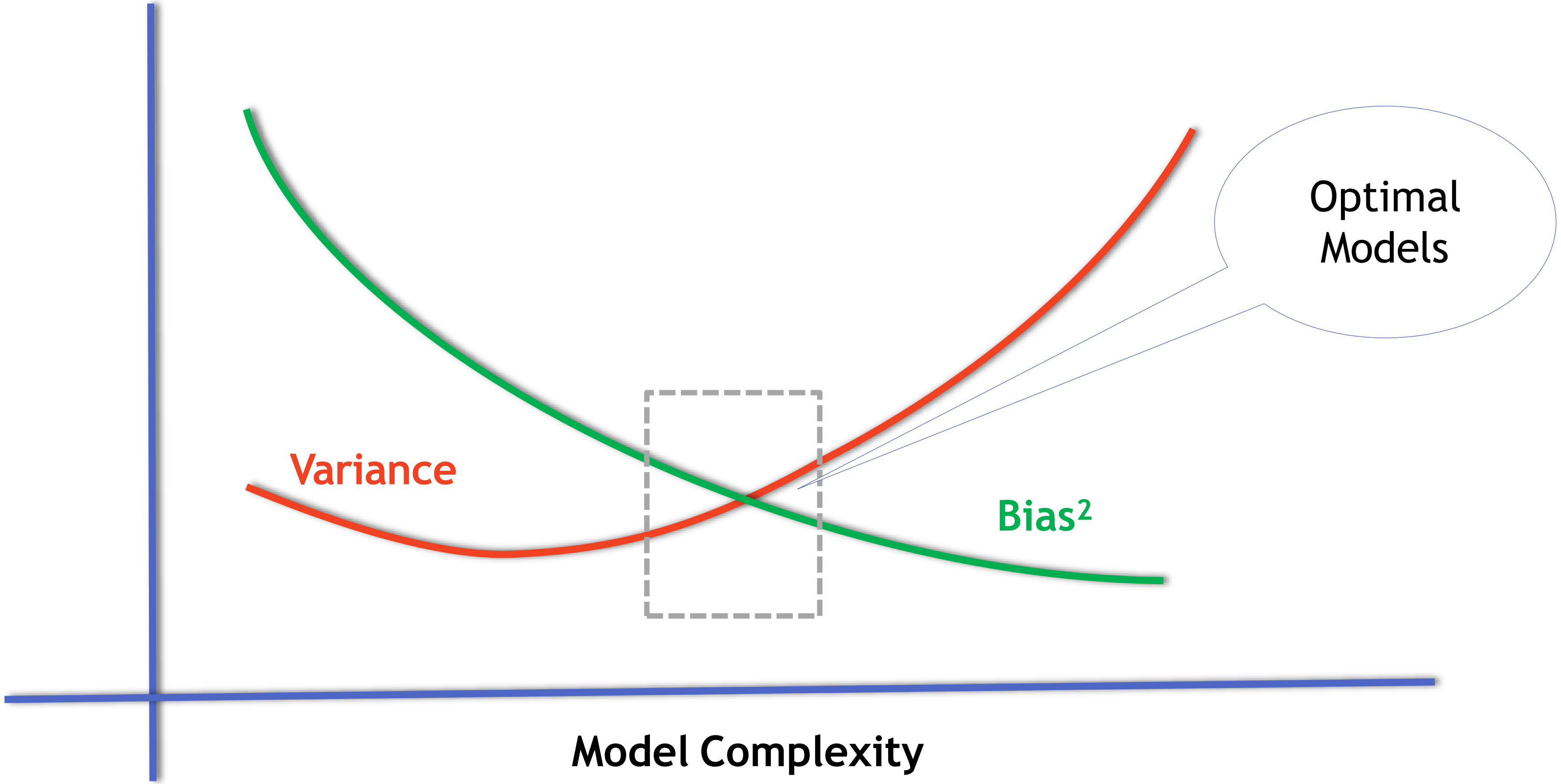
Overall Model Squared Error = Irreducible Error + Bias^2 + Variance

Bias-Variance Decomposition

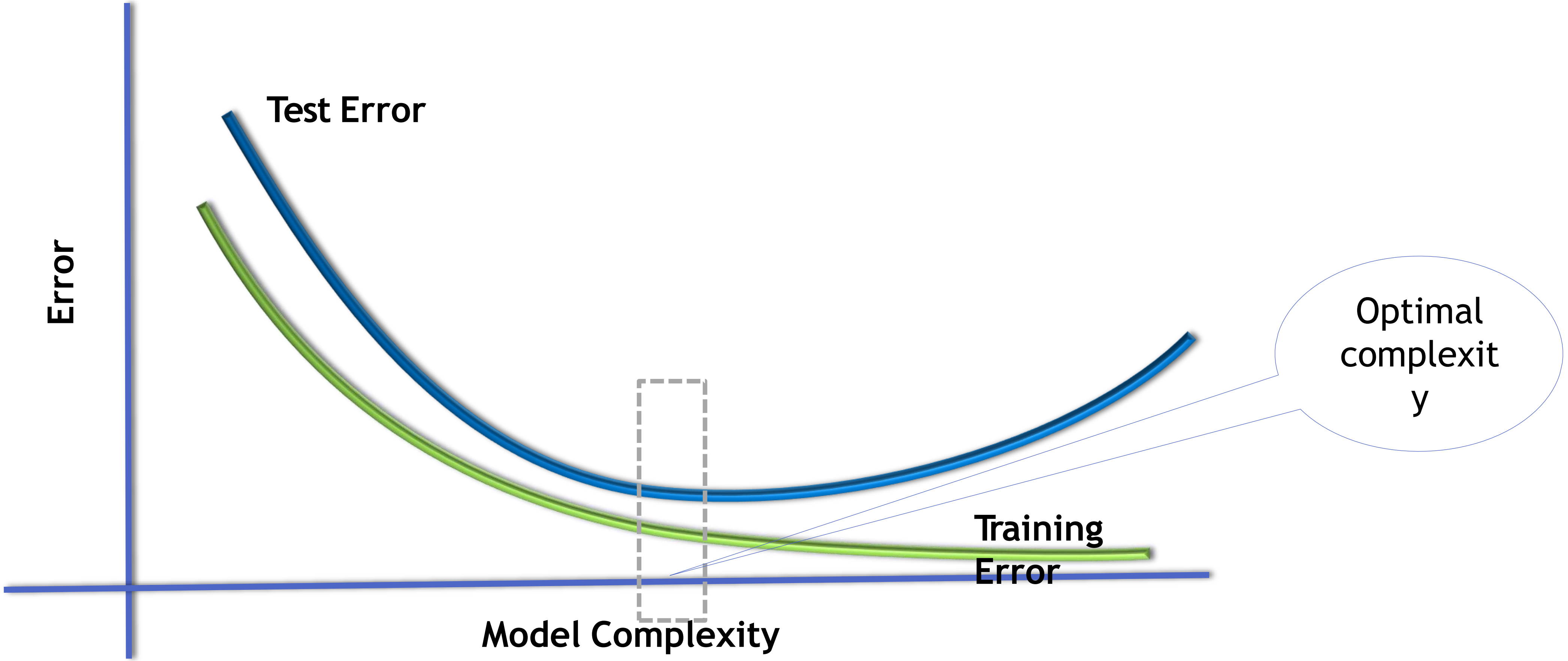
- Overall Model Squared Error = Irreducible Error + Bias^2 + Variance
- Overall error is made by bias and variance together
- High bias low variance, Low bias and high variance, both are bad for the overall accuracy of the model
- A good model need to have low bias and low variance or at least an optimal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

Choosing optimal model-Bias Variance Tradeoff

Bias Variance Tradeoff



Test and Training error



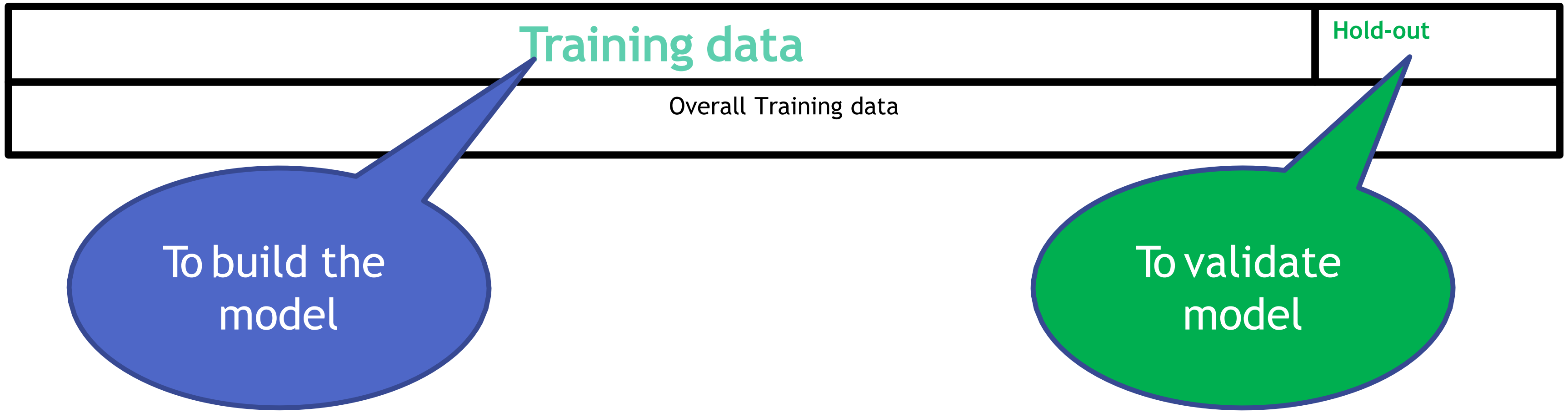
Choosing optimal model

- Unfortunately
  - There is no scientific method of choosing optimal model complexity that gives minimum test error.
  - Training error is not a good estimate of the test error.
  - There is always bias-variance tradeoff in choosing the appropriate complexity of the model.
  - We can use cross validation methods, boot strapping and bagging 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 Crossvalidation

- 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

Code: Holdout data Cross validation

```
#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


X= np.array(Fiber_df[features])
y = np.array(Fiber_df['active_cust'])

X_train, X_test, y_train, y_test =train_test_split(X,y, train_size =0.8)
#Defining tree parameters and training the tree
tree_CV =tree.DecisionTreeClassifier(criterion='gini',
                                     splitter='best',
                                     max_depth=20,
                                     min_samples_split=2,
                                     min_samples_leaf=1)
tree_CV.fit(X_train,y_train)

#Training score
tree_CV.score(X_train,y_train)

#Use the validation data to verify the error rate. Is the error rate on the training data and validation data same?
#Validation Accuracy on test data

tree_CV.score(X_test,y_test)
```

Training Data
0.9557499999999999

Test Data
0.8630499999999998

Code: Holdout data Cross validation

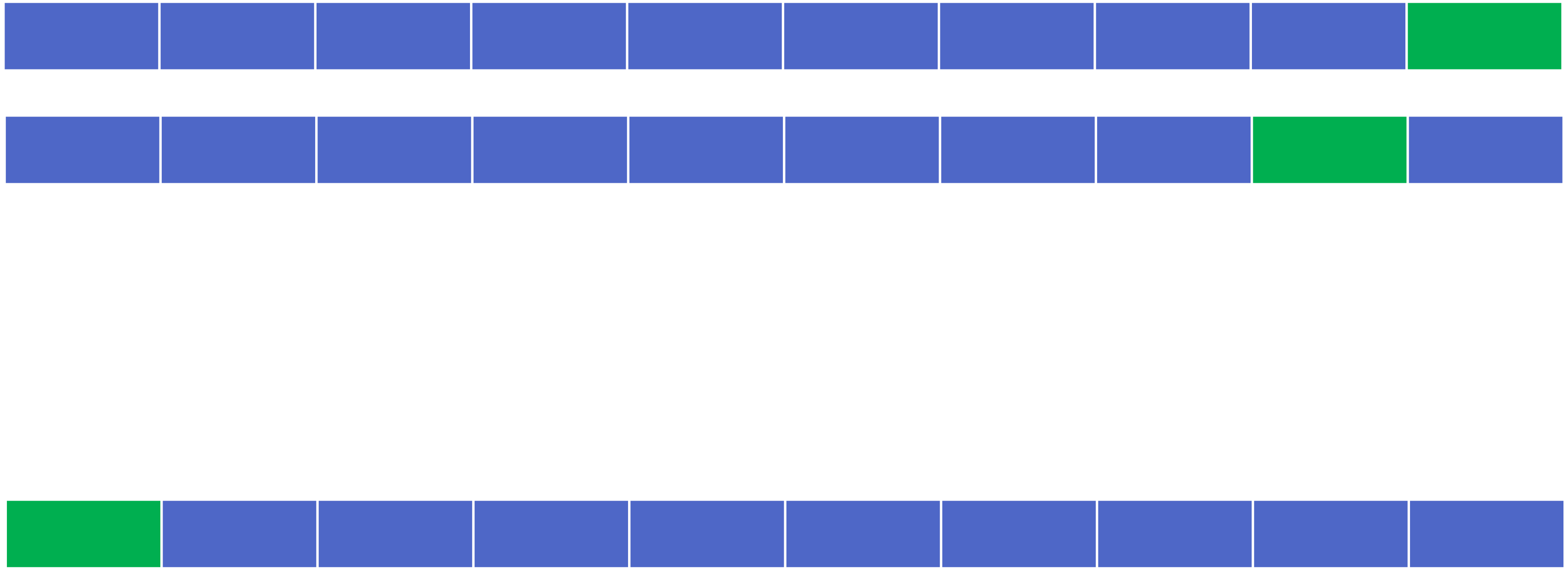
```
#Try to increase or reduce the complexity and choose the best model that performs well on training data as well as holdout data
#Improving the above model:
tree_CV1 = tree.DecisionTreeClassifier(criterion='gini',
                                       splitter='best',
                                       max_depth=10,
                                       min_samples_split=30,
                                       min_samples_leaf=30,
                                       max_leaf_nodes=30)
tree_CV1.fit(X_train,y_train)

#Training score of this pruned tree model
tree_CV1.score(X_train,y_train)
#Validation score of pruned tree model
tree_CV1.score(X_test,y_test)
```

```
Training Data
0.86003750000000001

Test Data
0.85845000000000005
```

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



K-fold - Validation

K-fold Cross Validation

- A generalization of cross validation.
- Divide the whole dataset into k equal parts
- Use kth 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

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?

Code K-fold Cross Validation

```
X= np.array(Fiber_df[features])
y = np.array(Fiber_df['active_cust'])

tree_KF = tree.DecisionTreeClassifier(criterion='gini',
                                      splitter='best',
                                      max_depth=30,
                                      min_samples_split =30,
                                      min_samples_leaf=30,
                                      max_leaf_nodes=60)
#Simple K-Fold cross validation. 10 folds.
from sklearn.cross_validation import KFold
kfold =KFold(len(Fiber_df), n_folds=10)

## Checking the accuracy of model on 10-folds
from sklearn import cross_validation
score10 =cross_validation.cross_val_score(tree_KF,X, y,cv=kfold)
score10
score10.mean()
```

10 Fold Output

```
[ 0.8358  0.703    0.6184 0.8047  0.8385
0.7995  0.7675  0.7507  0.7913   0.7206]
0.7630000000000001
```

Code K-fold Cross Validation

```
#Simple K-Fold cross validation. 20 folds.
kfold = KFold(len(Fiber_df), n_folds=20)

#Accuracy score of 20-fold model
score20 = cross_validation.cross_val_score(tree_KF,X, y,cv=kfold)
print(score20)
score20.mean()
```

20 Fold CV output

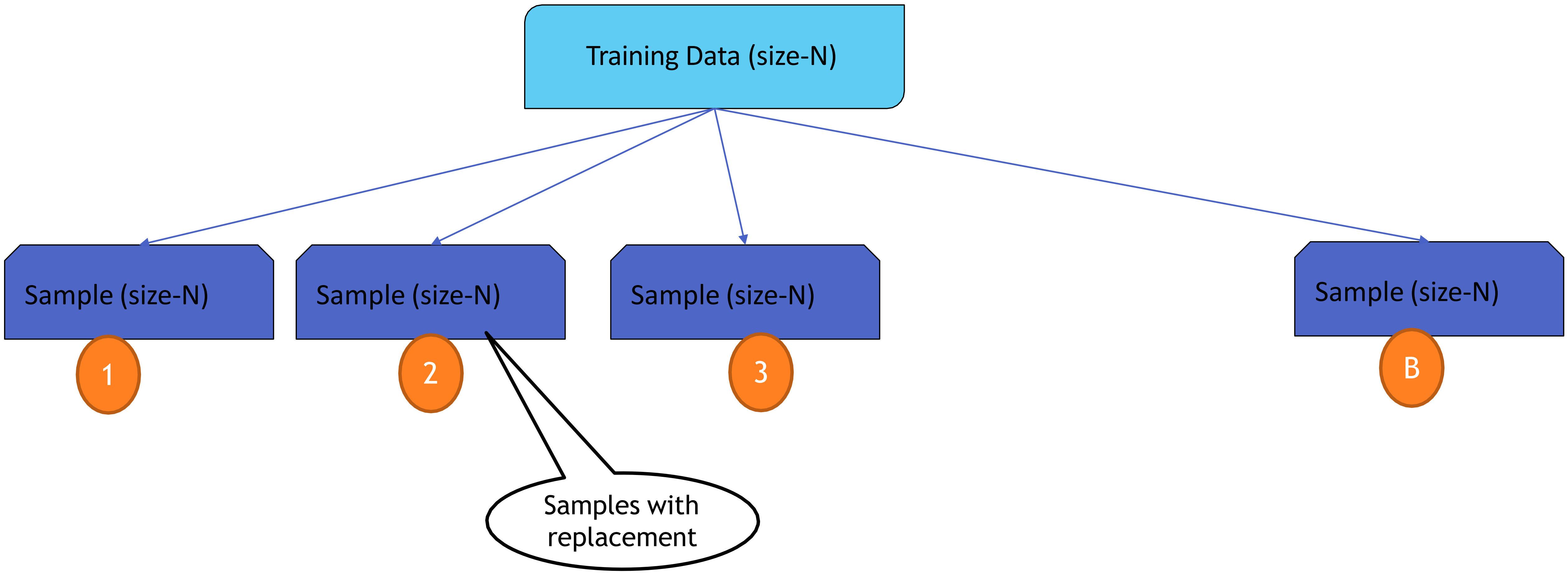
```
[ 0.9048  0.781   0.8288  0.612   0.283   0.6676  0.9226  0.7482  0.907   0.7866
0.6784  0.866   0.8788  0.9112  0.925   0.7318  0.9724  0.7502  0.6954  0.7456]

0.77981999999999996
```

Bootstrap Cross Validation

Bootstrap Methods

- Boot strapping is a powerful tool to get an idea on accuracy of the model and the test error
- Can estimate the likely future performance of a given modeling procedure, on new data not yet realized.
- 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 Method



Training Data (size-N)

Sample (size-N)

Sample (size-N)

Sample (size-N)

Sample (size-N)

Samples with replacement

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. We can even use the original training data as the test data for each of the models

LAB: Bootstrap Cross Validation

LAB: Bootstrap cross validation

- 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

```
tree_BS = tree.DecisionTreeClassifier(criterion='gini',
                                      splitter='best',
                                      max_depth=30,
                                      min_samples_split=30,
                                      min_samples_leaf=50,
                                      max_leaf_nodes=60)

# Defining the bootstrap variable for 10 random samples
bootstrap=cross_validation.ShuffleSplit(n=len(Fiber_df),
                                        n_iter=10,
                                        random_state=0)

###checking the error in the Boot Strap models###
BS_score = cross_validation.cross_val_score(tree_BS,X, y,cv=bootstrap)
BS_score

#Expected accuracy according to bootstrap validation
###checking the error in the Boot Strap models###
BS_score.mean()
```

```
Bootstrap output
0.8658, 0.8699, 0.8658, 0.8654,
0.8707, 0.8741, 0.8688, 0.8689,
0.8636, 0.8677

Mean
0.8680700
```

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
- R-square, Adj-R-square, Accuracy, ROC, AUC, AIC and BIC can be used to get an idea on training error
- Cross Validation and Boot strapping techniques give us an idea on test error
- Choose the model based on the combination of AIC, Cross Validation and Boot strapping results
- Bootstrap is widely used in ensemble models & random forests.

References

- Hastie, Tibshirani and Friedman .The Elements of Statistical Learning (2nd edition, 2009).
- http://scott.fortmann-roe.com/docs/BiasVariance.html