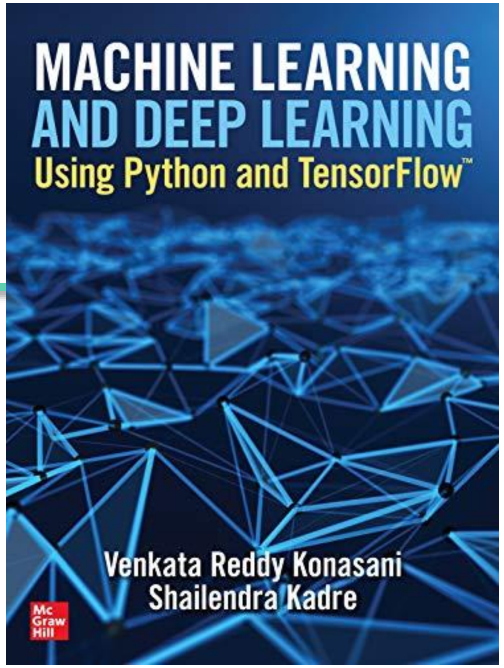
Model Selection and Cross Validation

Venkat Reddy

Chapter 5 in the book



Contents

- How to validate a model?
- Sensitivity, Specificity, Recall and Precision
  Types of data
- Types of errors
- The problem of over fitting
- The problem of under fitting
- Bias Variance Tradeoff
- Cross validation

Model Building Life Cycle

Background and Objective	Data Exploration	Preparing data for analysis	Building the model	Validating the model	Deployment
Business Objective	Collect data	Validate data	Select the right model	Intime validation	Deploy model
Set Goals	Explore data	Outlier treatment	Variable selection	Out of time validation	Maintenance of model
Project Plan	Basic Summary	Missing value treatment	Model building and finetuning	Model finetuning	Model monitoring
Budget & Resources	Identify outliers and missing values	Clean the data	Model iterations		
		Prepare data for Analysis			

Model Building Life Cycle

Background and Objective	Data Exploration	Preparing data for analysis	Building the model	Validating the model	Deployment
Business Objective	Collect data	Validate data	Select the right model	Intime validation	Deploy model
Set Goals	Explore data	Outlier treatment	Variable selection	Out of time validation	Maintenance of model
Project Plan	Basic Summary	Missing value treatment	Model building and finetuning	Model finetuning	Model monitoring
Budget & Resources	Identify outliers and missing values	Clean the data	Model iterations		
		Prepare data for Analysis			

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 R-squared for regression, confusion matrix and accuracy for classification. There are many more validation and model accuracy metrics for classification models

statinfer

Additional Validation metrics for Regression Problems

Validation metrics for Regression Problems

Error at one point	$y_i - \hat{y}_i$
Sum of squares of errors(SSE)	$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
Mean Absolute Deviation(MAD)	$\sum_{i=1}^{n}\frac{\lvert y_i - \hat{y}_i\rvert}{n}$
Mean absolute percent error (MAPE)	$\frac{100}{n}\sum_{i=1}^{n}\frac{\lvert y_i - \hat{y}_i\rvert}{y_i}$
Mean Square Error(MSE)	$\sum_{i=1}^{n}\frac{(y_i - \hat{y}_i)^2}{n}$
Root Mean Square Error(RMSE)	$\sqrt{MSE}$

statinfer

Additional Validation metrics for Classification Problems

Classification-Validation measures

- Confusion matrix, Specificity, Sensitivity
- Precision, Recall
- F1 Score
- 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

99.999% Accuracy ..

- 99.999% accuracy on Train data
- 99.999% accuracy on Test data
- Is it a good model ?

Predicting a Bomb

- Model Predicts the bomb
- Imagine we applied the model on 100,000 cars
  - 99,999 cars have no bomb - Class0
  - One car a bomb – Class1

- Our model predicted every car is safe, none of the cars have bomb.
  - 100,000 are predicted as class 0

Actual values

Row_id								Y Actual	Y Predicted
1								0	
2								0	
3								0	
4								0	
5								0	
6								0	
7								0	
8								0	
9								0	
.								0	
.								0	
.								0	
100,000								1	

Actual vs predicted

Row_id								Y Actual	Y Predicted
1								0	0
2								0	0
3								0	0
4								0	0
5								0	0
6								0	0
7								0	0
8								0	0
9								0	0
.								0	0
.								0	0
.								0	0
100,000								1	0

statinfer

Model Accuracy

		Predicted Classes	
		0- No Bomb	1 - Bomb
Actual Classes	0 - No Bomb		
	1 - Bomb		

Model Accuracy

		Predicted Classes	
		0- No Bomb	1 - Bomb
Actual Classes	0 - No Bomb	99,999	0
	1 - Bomb	1	0

Class wise Accuracy

statinfer

Predicted Classes

Actual Classes	0- No Bomb	1 - Bomb	
0 - No Bomb	99,999	0	100%
1 - Bomb	1	0	0%

Sensitivity and Specificity

Classification Table

Sensitivity and Specificity are derived from confusion matrix

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
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

		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

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. **Rejected a Loan of 100,000**	False Negatives(FN) Actual customer is bad and model is predicting them as good **Issued a loan of 100,000**	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

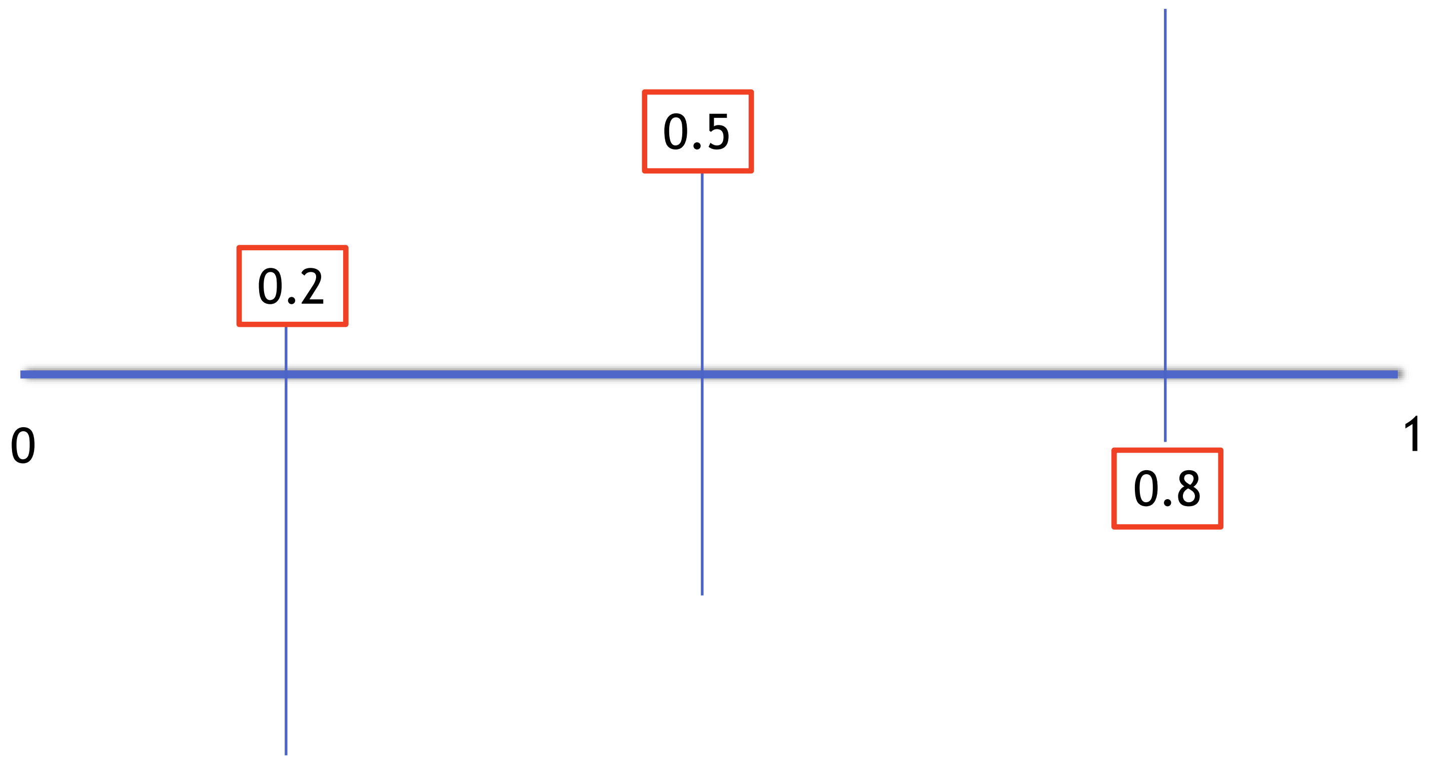
- There are some cases where Sensitivity is important and need to be near to 1
- There are business cases where Specificity is important and need to be near to 1
- We need to understand the business problem and decide the importance of Sensitivity and Specificity

Calculating Sensitivity and Specificity

LAB- Sensitivity and Specificity

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

The Threshold



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

Precision and Recall

Actual Classes \ Predicted 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
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 and Recall

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

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

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

Precision and Recall – Calculated for each class

statinfer

		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
	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

$$Class\ 0\ Recall = \frac{TP}{TP + FN}$$

$$Class\ 1\ Recall = \frac{TN}{FP + TN}$$

$$Class\ 0\ Precision = \frac{TP}{TP + FP}$$

$$Class\ 1\ Precision = \frac{TN}{FN + TN}$$

statinfer

F1 – Score for a class

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

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

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

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

statinfer

F1 – Score for a class

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

$$F1 - Score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

Predicted Classes

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

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

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

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

LAB - Precision, Recall and F1 Score

- Calculate Precision, Recall and F1 score for both the classes.

statinfer

ROC Curve

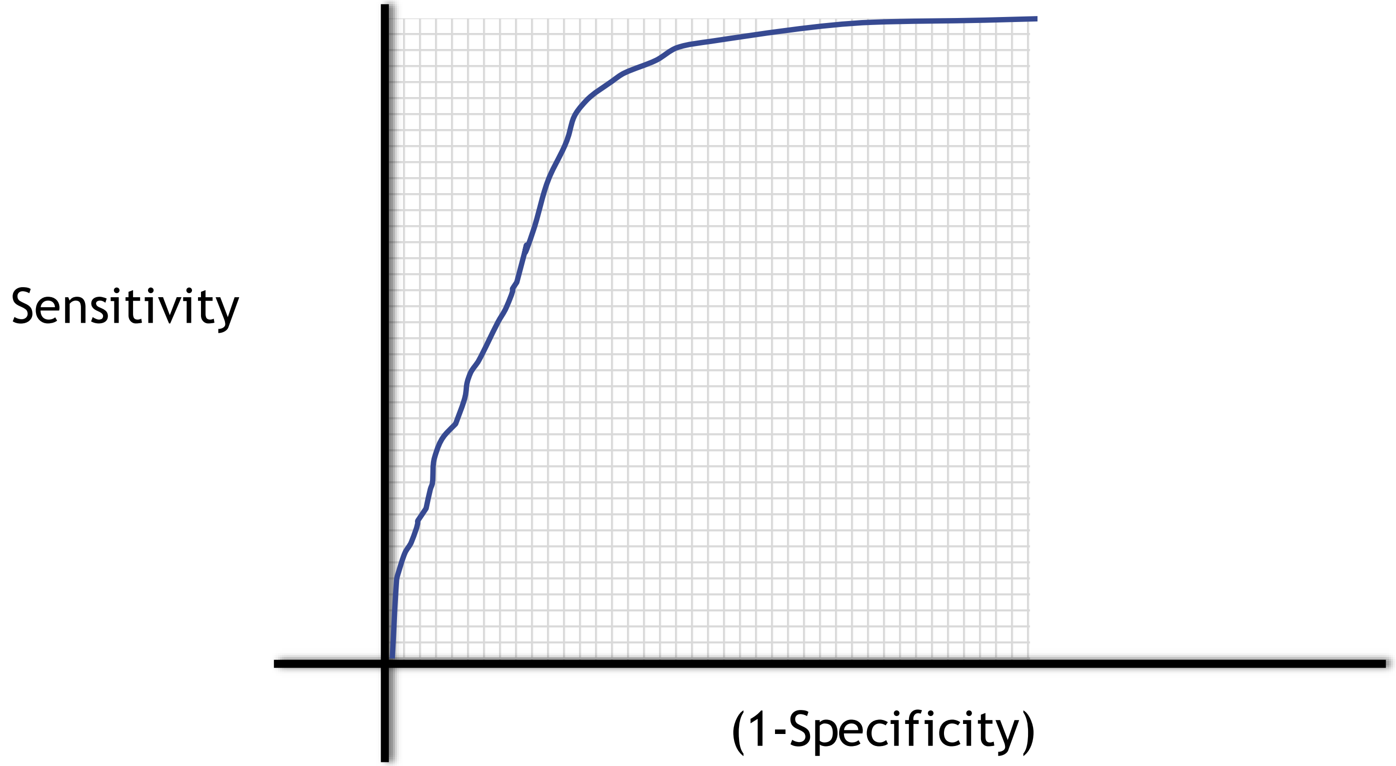
Changing the threshold

- Does changing the threshold changes the Accuracy ?
- Does changing the threshold changes the Sensitivity?
- Does changing the threshold changes the Specificity?

- How to choose the right threshold ?

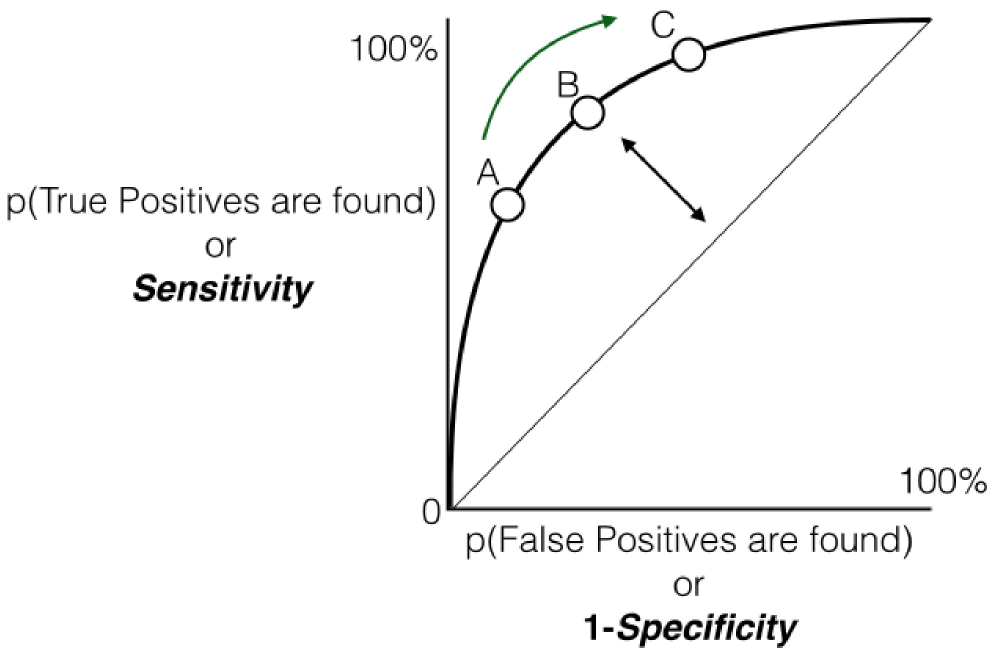
ROC Curve

- Consider All thresholds between [0-1]
  - Calculate Accurate predictions rate - Sensitivity (True Positive Rate)
  - Calculate the corresponding error rate – (1-Specificity) (False Positive Rate)

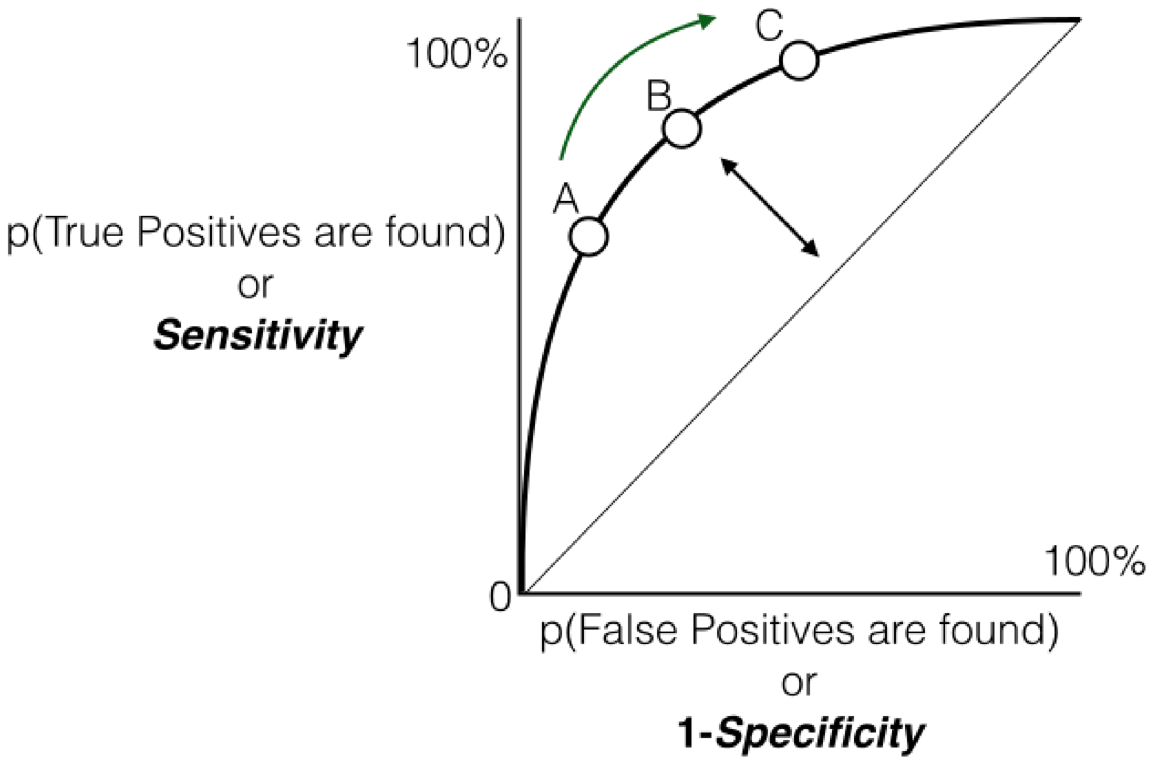


ROC Curve

- 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?

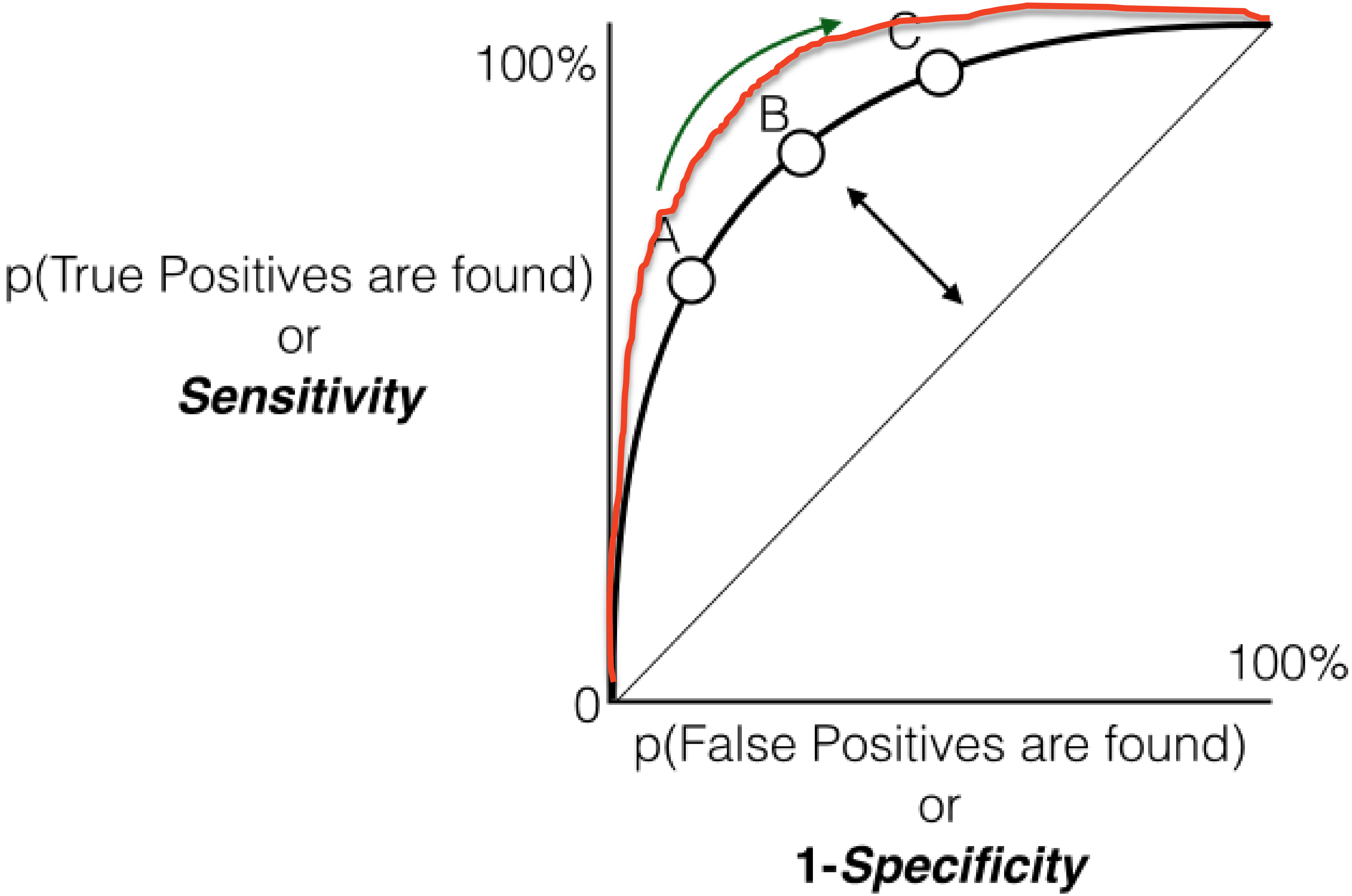


ROC Curve - Interpretation



- 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

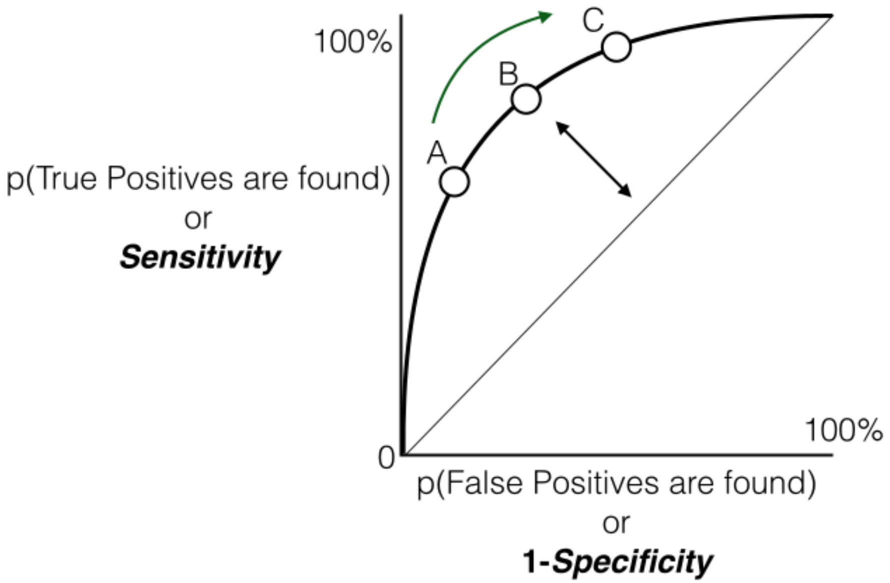
ROC Comparison for two models



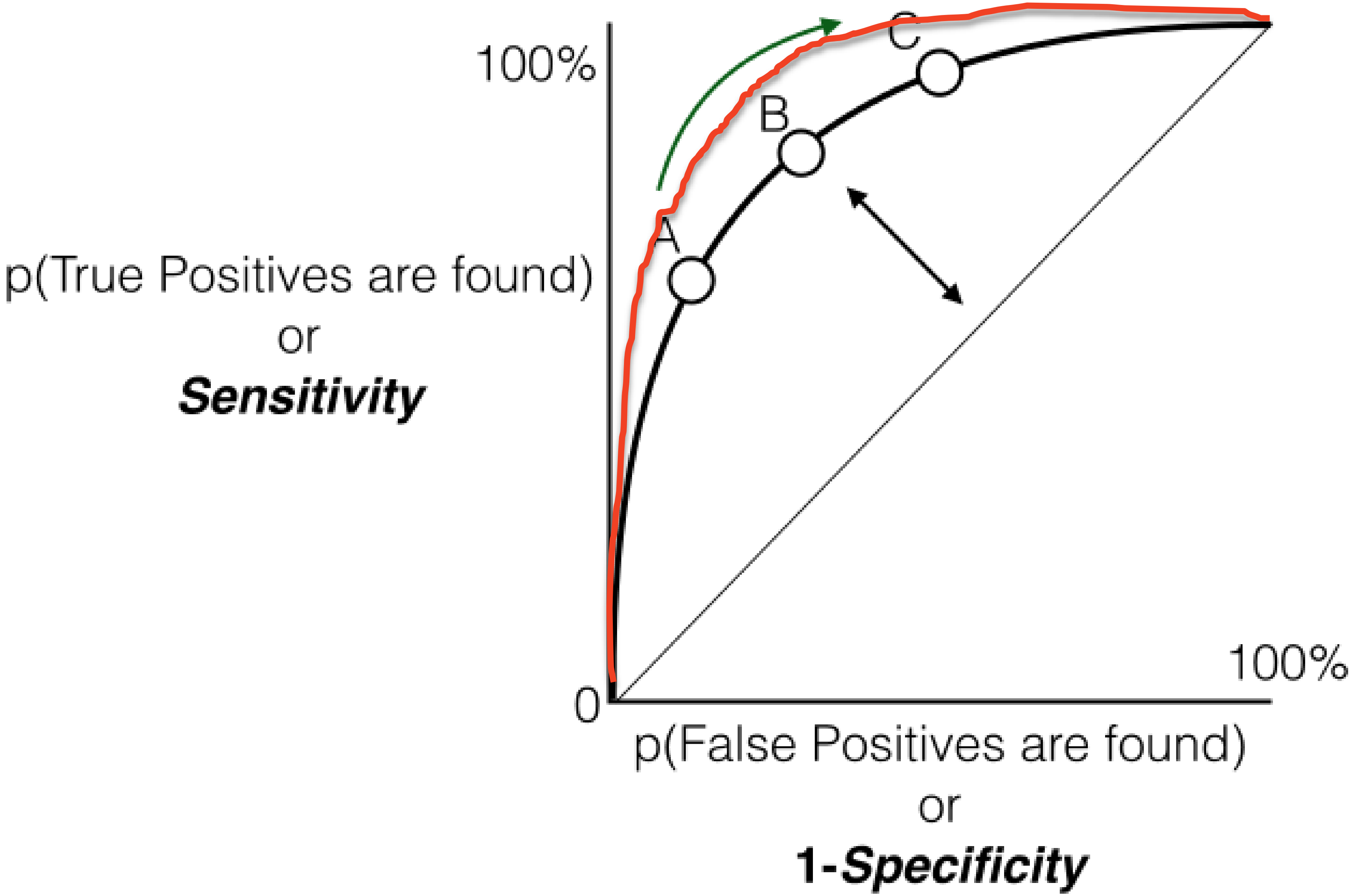
AUC

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



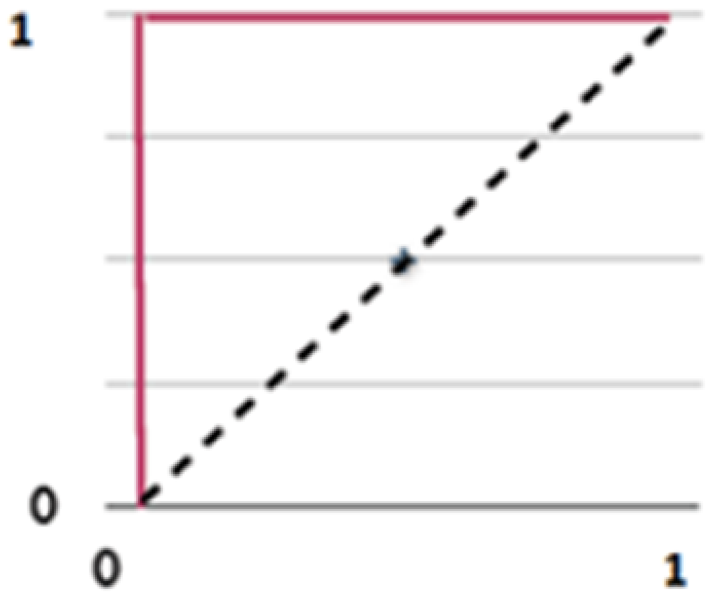
AUC Comparison for two models



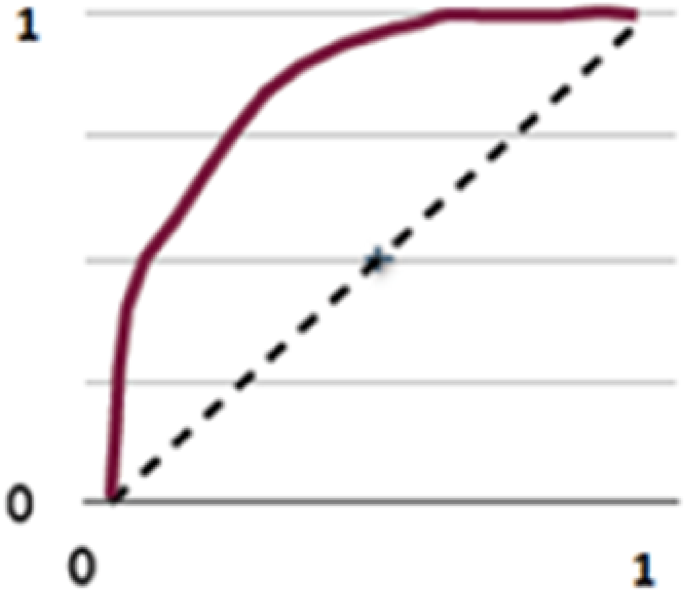
AUC

- AUC is near to 1 for a good model

AUC=1



AUC=0.82



ROC and AUC Calculation

Code: ROC and AUC

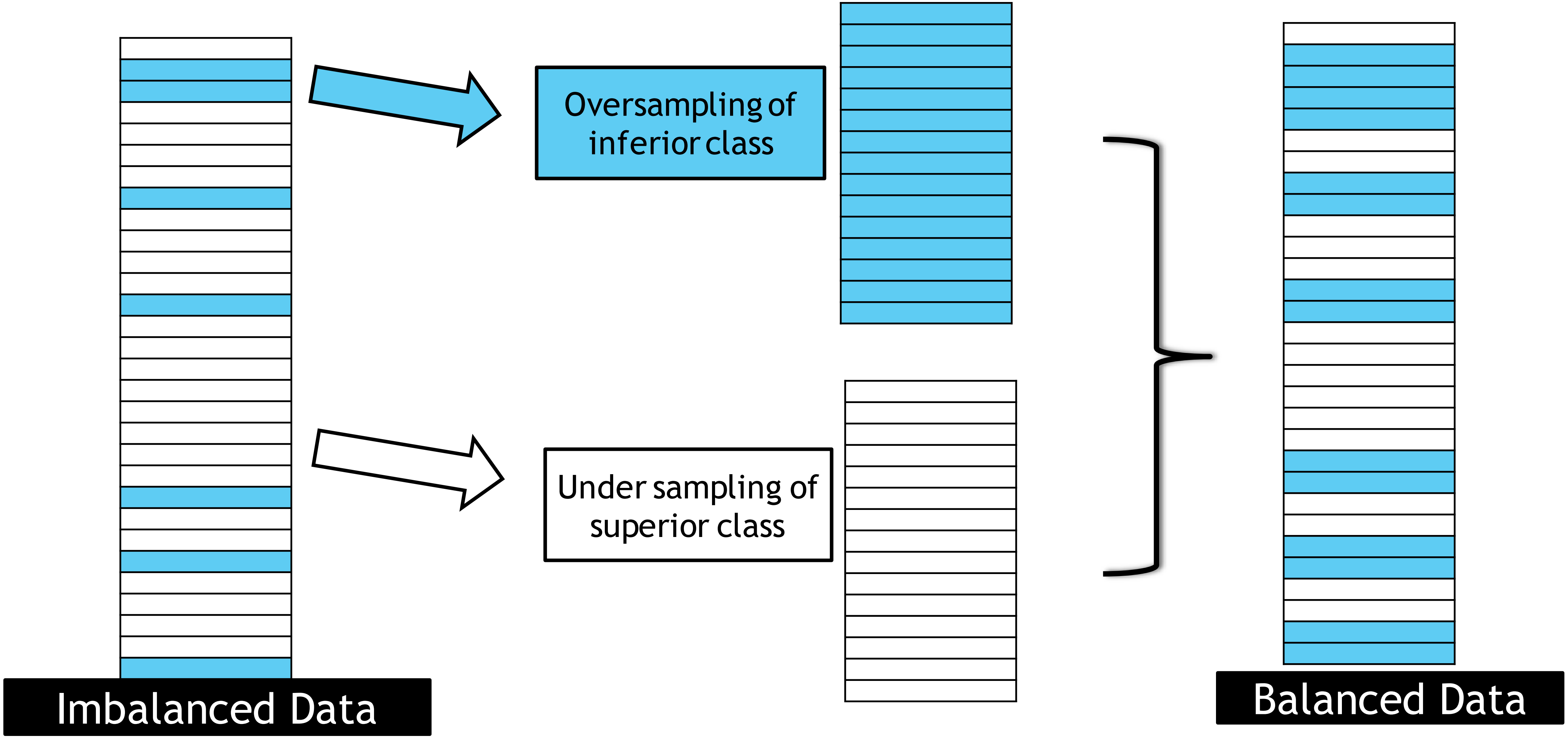
```
false_positive_rate, true_positive_rate, thresholds = roc_curve(actual, predictions)

plt.title('ROC Curve')
#Drawing ROC Curve
plt.plot(false_positive_rate, true_positive_rate)
# Drawing a line at y=1
plt.axvline(x=1, ymin=0, ymax=1, color='g', linewidth=3)
#X and Y  Axis Limits
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
# Labels
plt.ylabel('True Positive Rate(Sensitivity)')
plt.xlabel('False Positive Rate(Specificity)')
plt.show()

roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

Handling Class Imbalance

Oversampling and Under Sampling



Code- Oversampling and Under sampling

```
print("Actual Data :", loans.shape)

#Frequency count on target column
freq=loans['SeriousDlqin2yrs'].value_counts()
print(freq)
print((freq/freq.sum())*100)

#Classwise data
credit_risk_class0 = loans[loans['SeriousDlqin2yrs'] == 0]
credit_risk_class1 = loans[loans['SeriousDlqin2yrs'] == 1]

print("Class0 Actual :", credit_risk_class0.shape)
print("Class1 Actual  :", credit_risk_class1.shape)
```

```
Actual Data : (150000, 14)
0    139974
1     10026
Name: SeriousDlqin2yrs, dtype: int64
0    93.316
1     6.684
Name: SeriousDlqin2yrs, dtype: float64
Class0 Actual : (139974, 14)
Class1 Actual  : (10026, 14)
```

Code- Oversampling and Under sampling

```
##Undersampling of class-0
## Consider half of class-0
credit_risk_class0_under = credit_risk_class0.sample(int(0.5*len(credit_risk_class0)))
print("Class0 Undersample :", credit_risk_class0_under.shape)

##Oversampling of Class-1
# Lets increase the size by four times
credit_risk_class1_over = credit_risk_class1.sample(4*len(credit_risk_class1),replace=True)
print("Class1 Oversample :", credit_risk_class1_over.shape)

#Concatenate to create the final balanced data
credit_risk_balanced=pd.concat([credit_risk_class0_under,credit_risk_class1_over])
print("Final Balannced Data :", credit_risk_balanced.shape)
```

Code- Oversampling and Under sampling

```
Class0 Undersample : (69987, 14)
Class1 Oversample : (40104, 14)
Final Balannced Data : (110091, 14)
0    69987
1    40104
Name: SeriousDlqin2yrs, dtype: int64
0    63.571954
1    36.428046
Name: SeriousDlqin2yrs, dtype: float64
```

Code- New model and results

```
risk_model=sm.logit(model_formula, data=credit_risk_balanced)
results=risk_model.fit()
print(results.summary())
```

```
Confusion Matrix :
 [[63703  6284]
 [17679 22425]]
Accuracy :  0.7823346140919785
Sensitivity :  0.9102118964950634
Specificity :  0.5591711549970078
```

```
Accuracy :  0.7823346140919785
Precision_Class0 :  0.7827652306406823
Recall_Class0 :  0.9102118964950634
F1_Class0 :  0.8416914956166719
Precision_Class1 :  0.78111936396252
Recall_Class1 :  0.5591711549970078
F1_Class1 :  0.6517663813523606
```

statinfer

Synthetic Minority Oversampling Technique

SMOTE

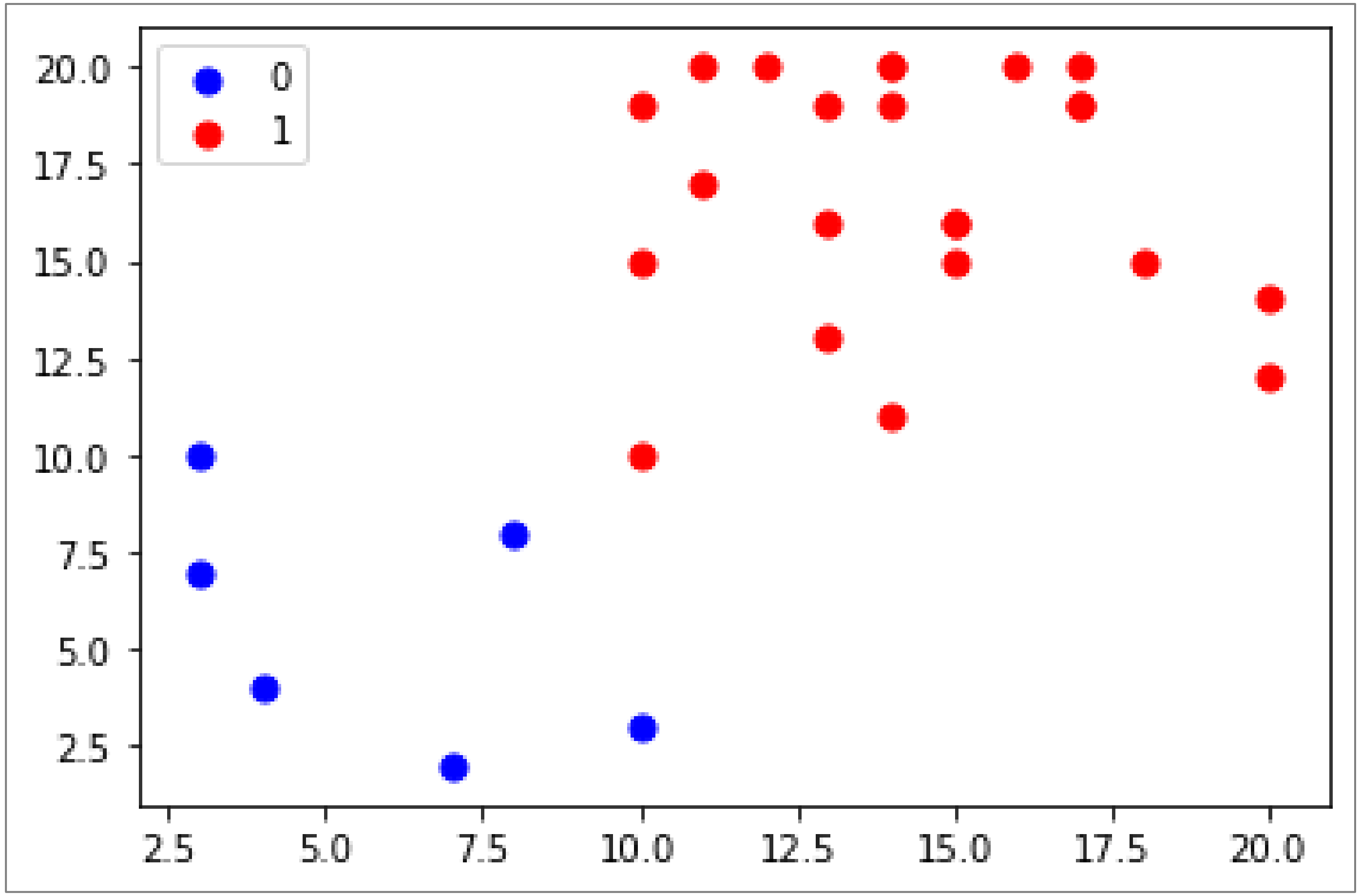
Creating Synthetic Samples

Income	Monthly Utilization	Defaulter
75,000	60%	0
80,000	65%	0
77,500	62.5%	0

Synthetic sample

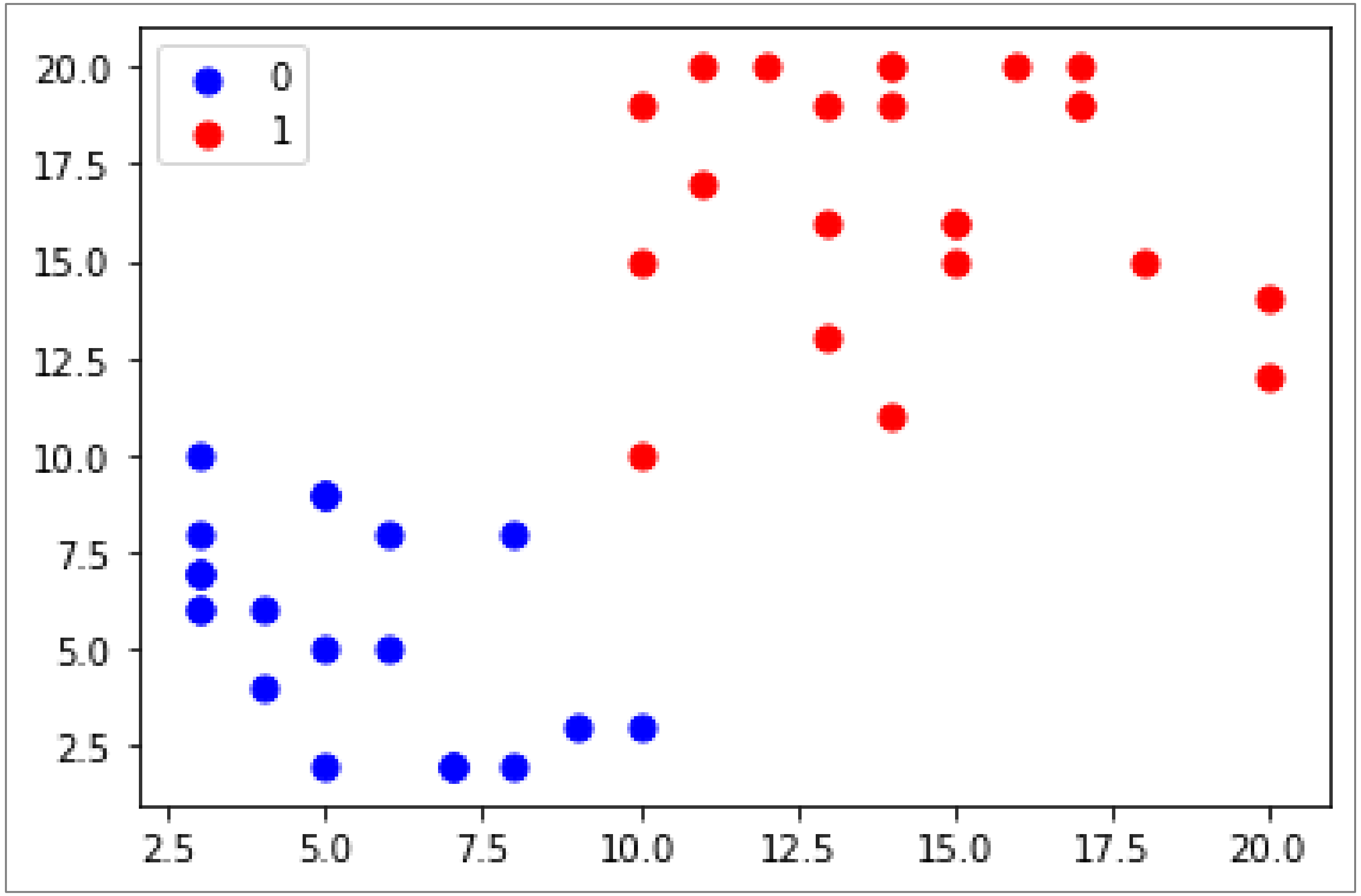
Synthetic Minority Oversampling Technique

- SMOTE technique creates synthetic samples based on the data in minor class

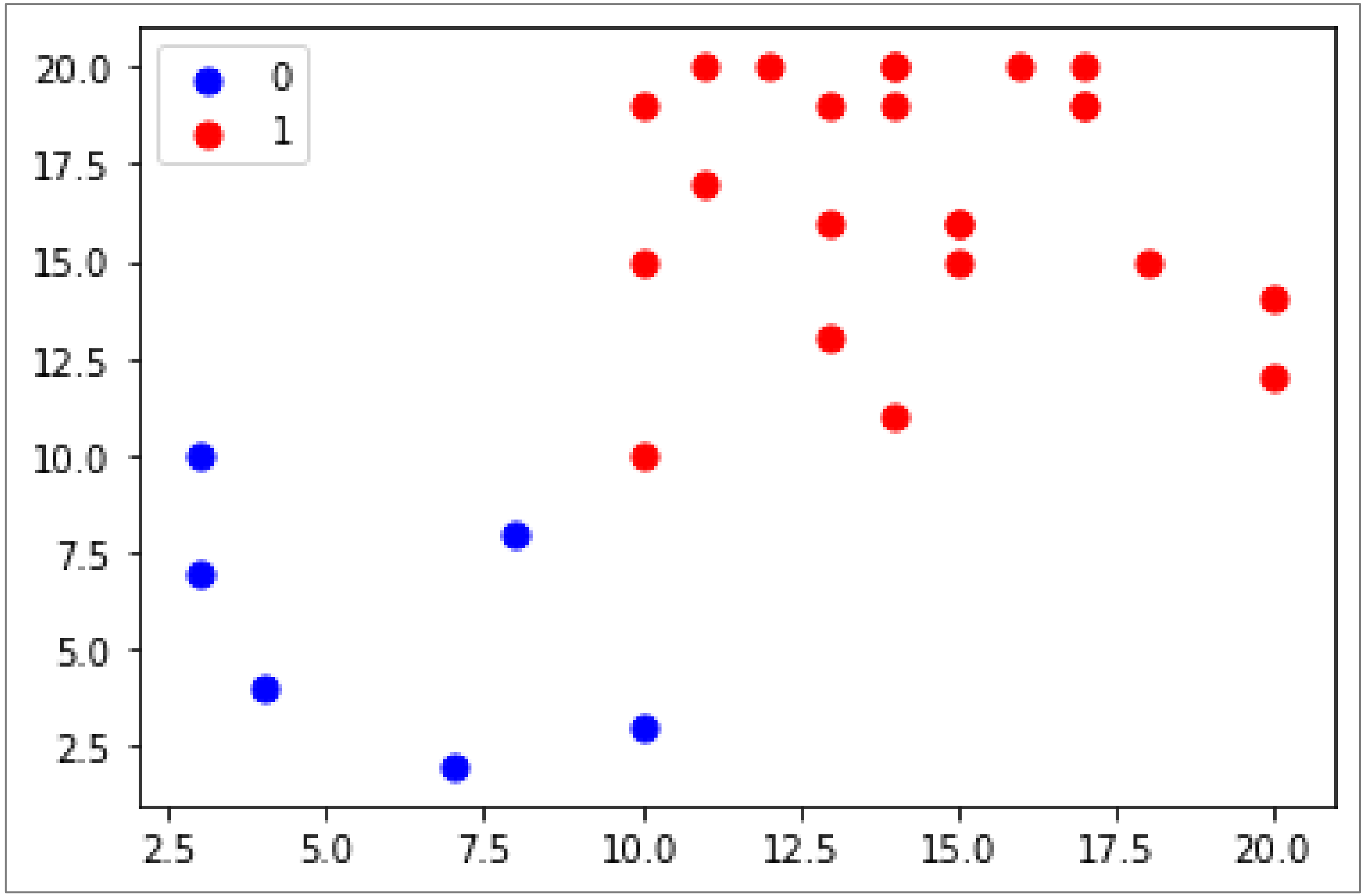


Synthetic Minority Oversampling Technique

- Data creating synthetic samples

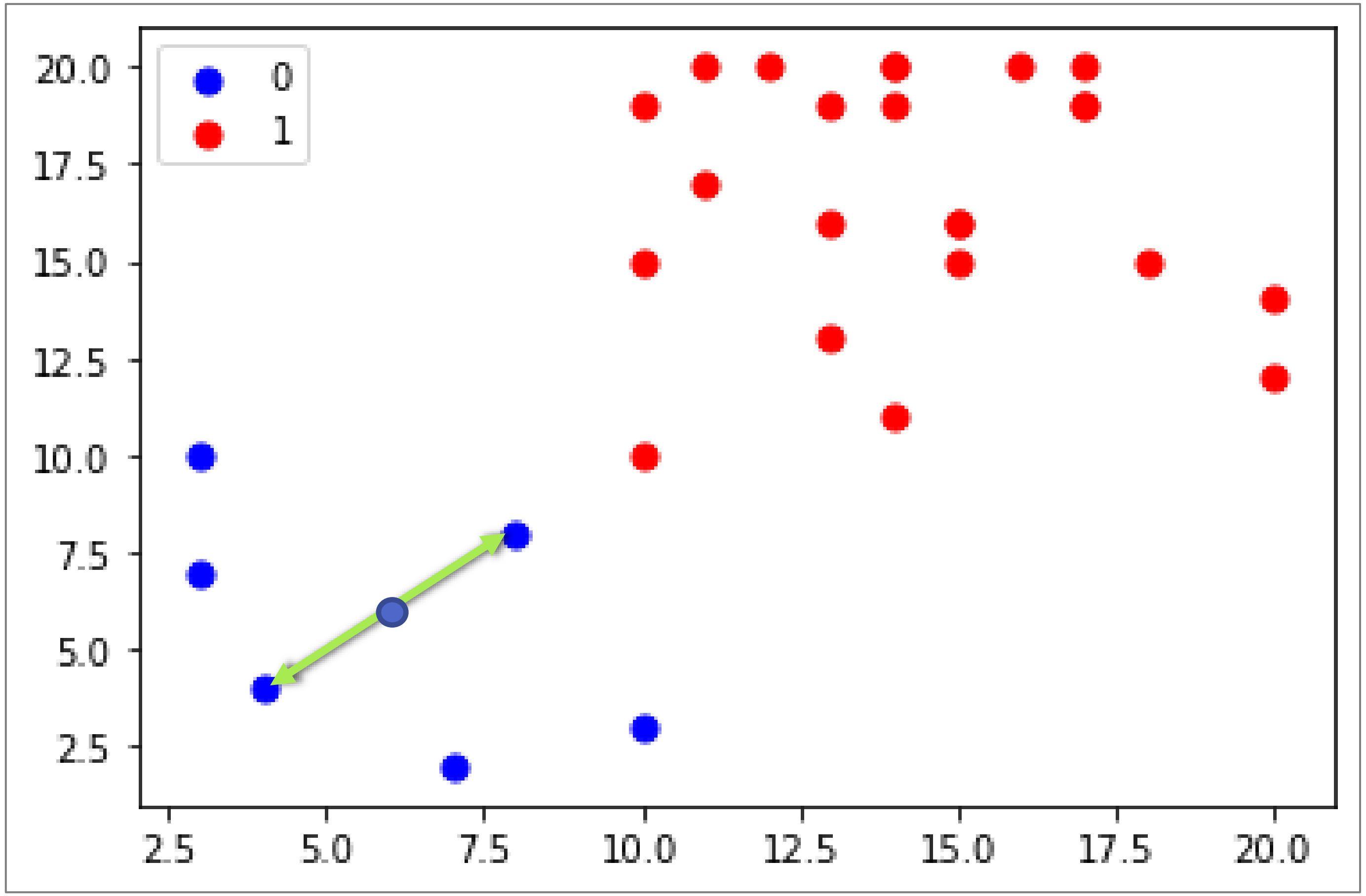


SMOTE- Theory Behind it



1. Take a data point from the minority class
2. Select the K- nearest neighbours (k=5 by default). Nearest neighbours are selected based on Euclidian distance
3. Randomly select a point from the nearest neighbours set
4. Interpolate and cerate a new synthetic data point

SMOTE- Theory Behind it



1. Take a data point from the minority class
2. Select the K- nearest neighbours (k=5 by default). Nearest neighbours are selected based on Euclidian distance
3. Randomly select a point from the nearest neighbours set
4. Interpolate and cerate a new synthetic data point

Code - SMOTE

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state = 2)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train.ravel())

import collections
print("Before SMOTE", collections.Counter(y_train))
print("After SMOTE", collections.Counter(y_train_smote))
```

```
Before SMOTE Counter({0: 139974, 1: 10026})
After SMOTE Counter({0: 139974, 1: 139974})
```

Minority class size is increased to match the size of majority class

Code - SMOTE

Minority class size is increased match 60% of majority class

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy=0.6, random_state = 2)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train.ravel())

import collections
print("Before SMOTE", collections.Counter(y_train))
print("After SMOTE", collections.Counter(y_train_smote))
```

```
Before SMOTE Counter({0: 139974, 1: 10026})
After SMOTE Counter({0: 139974, 1: 83984})
```

Minority class size is increased match 60% of majority class

Result after SMOTE

```
from sklearn.metrics import classification_report
print(classification_report(credit_risk_balanced["SeriousDlqin2yrs"],predicted_class1))
```

```
              precision    recall  f1-score   support

           0       0.77      0.86      0.81    139974
           1       0.71      0.58      0.64     83984

    accuracy                           0.75    223958
   macro avg       0.74      0.72      0.73    223958
weighted avg       0.75      0.75      0.75    223958
```

statinfer

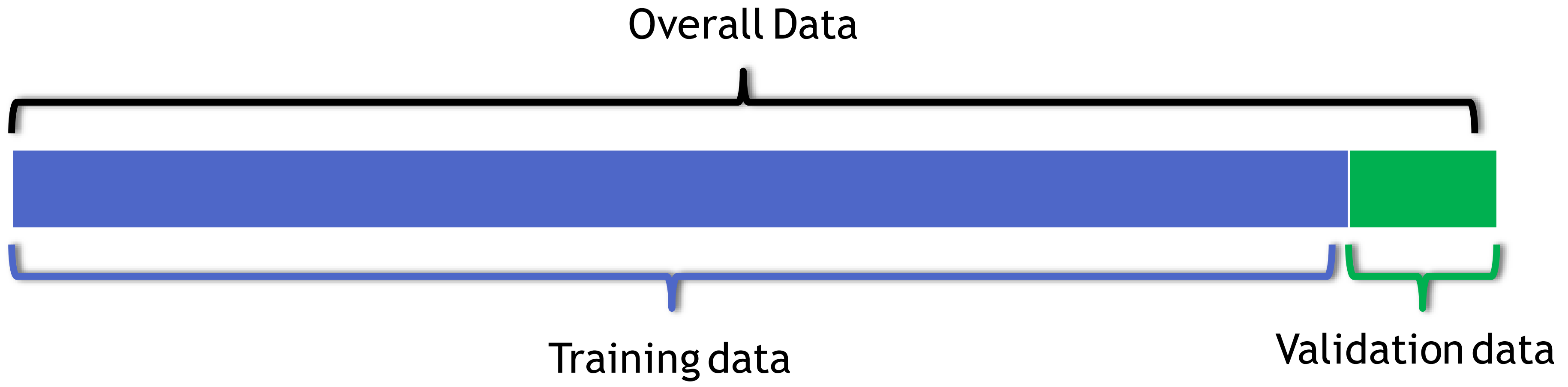
Different type of datasets and errors

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

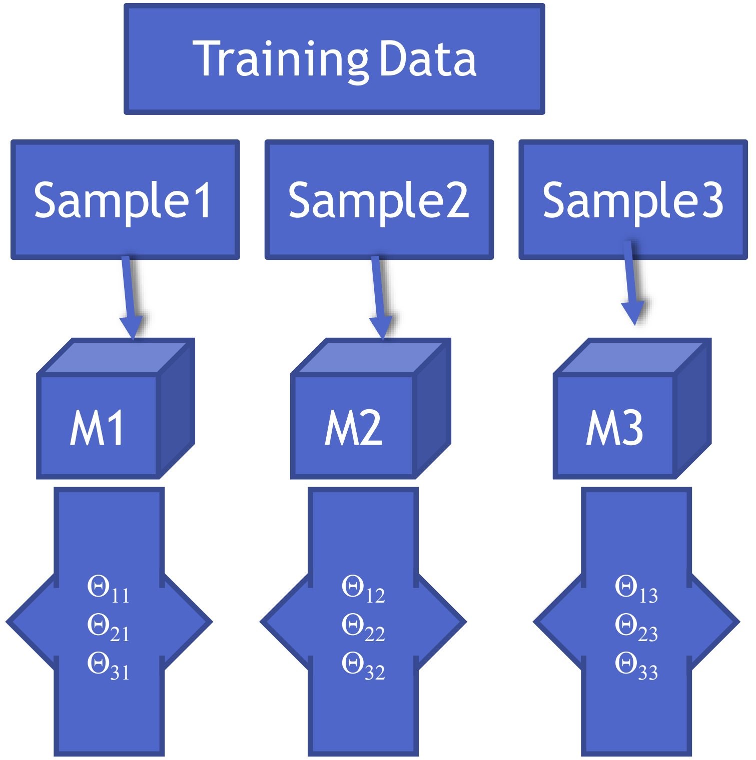
- The training error
  - The error on training dataset
  - In-time error
  - Error on the known data
  - Can be reduced while building the model
- The test error
  - The error that matters
  - Out-of-time error
  - The error on unknown/new dataset.

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



The problem of over fitting

- 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
  - 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

```
features = list(Fiber_df.drop(['active_cust'],1).columns)
X = np.array(Fiber_df[features])
y = np.array(Fiber_df['active_cust'])

#Splitting the dataset into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size = 0.8
#Build the best model(1% error) model on training data.
tree_model = tree.DecisionTreeClassifier()
tree_model.fit(X_train,y_train)

train_acc= tree_model.score(X_train,y_train)
print("train_accuracy", train_acc)

test_acc= tree_model.score(X_test,y_test)
print("test_accuracy", test_acc)
```

```
train_accuracy 0.997275
test_accuracy 0.84225
```

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.

Code: Model with huge Bias

```
tree_model = tree.DecisionTreeClassifier(max_depth=1)
tree_model.fit(X_train,y_train)

train_acc= tree_model.score(X_train,y_train)
print("train_accuracy", train_acc)

test_acc= tree_model.score(X_test,y_test)
print("test_accuracy", test_acc)
```

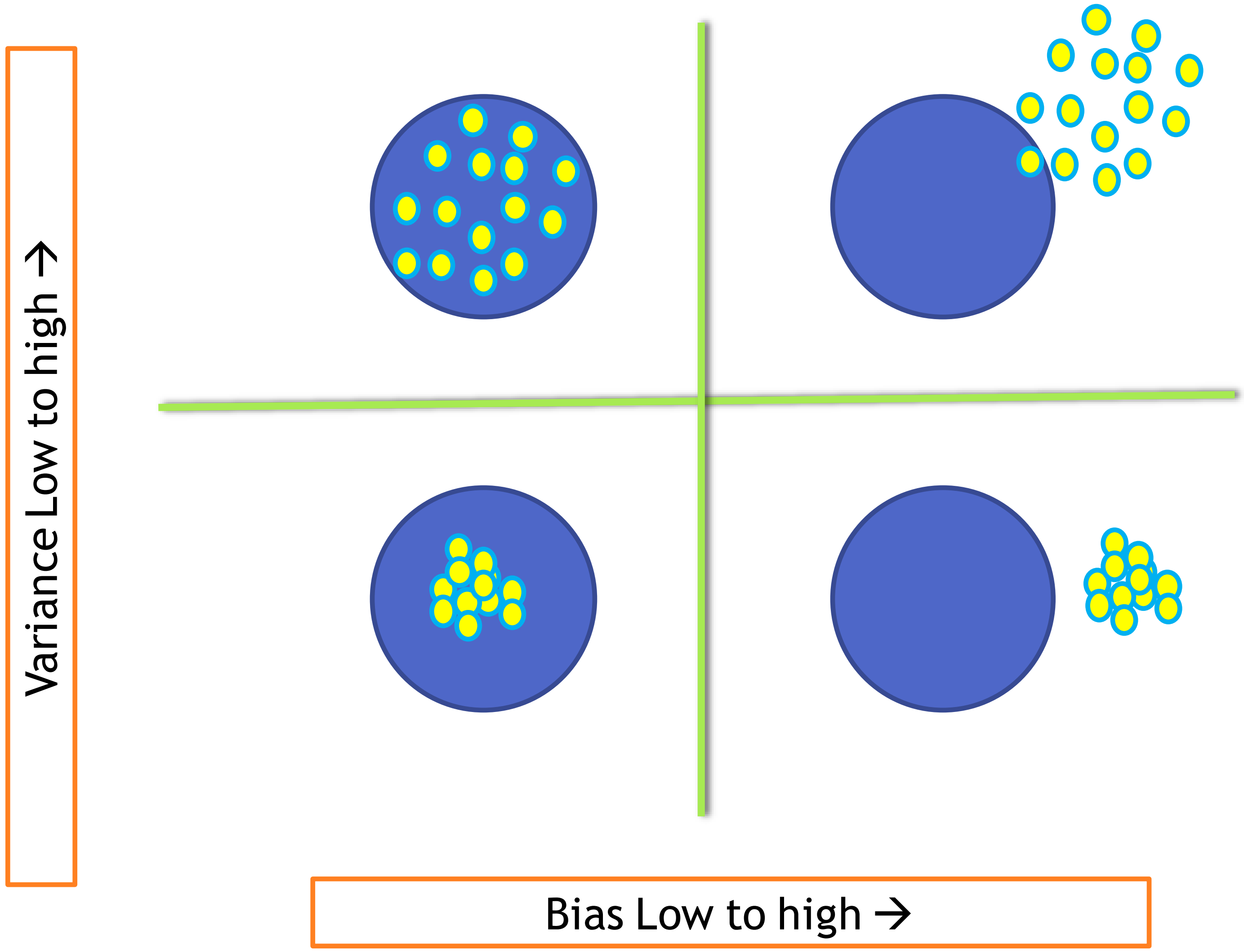
```
train_accuracy 0.6833875
test_accuracy 0.6814
```

Bias and Variance Tradeoff

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

The Bias-Variance Decomposition

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

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

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

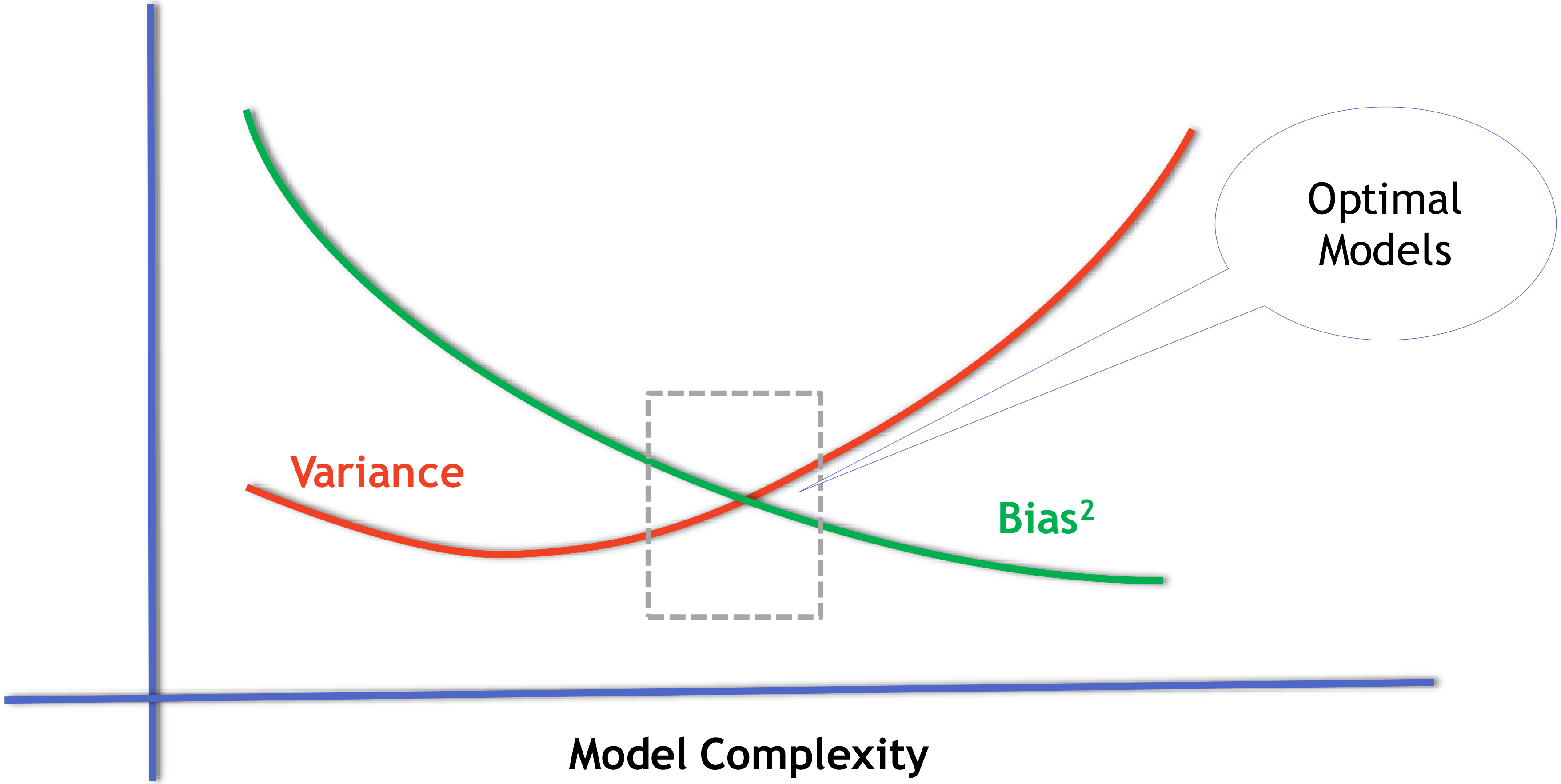
Overall Model Squared Error = Irreducible Error + Bias^2 + Variance

Bias-Variance Decomposition

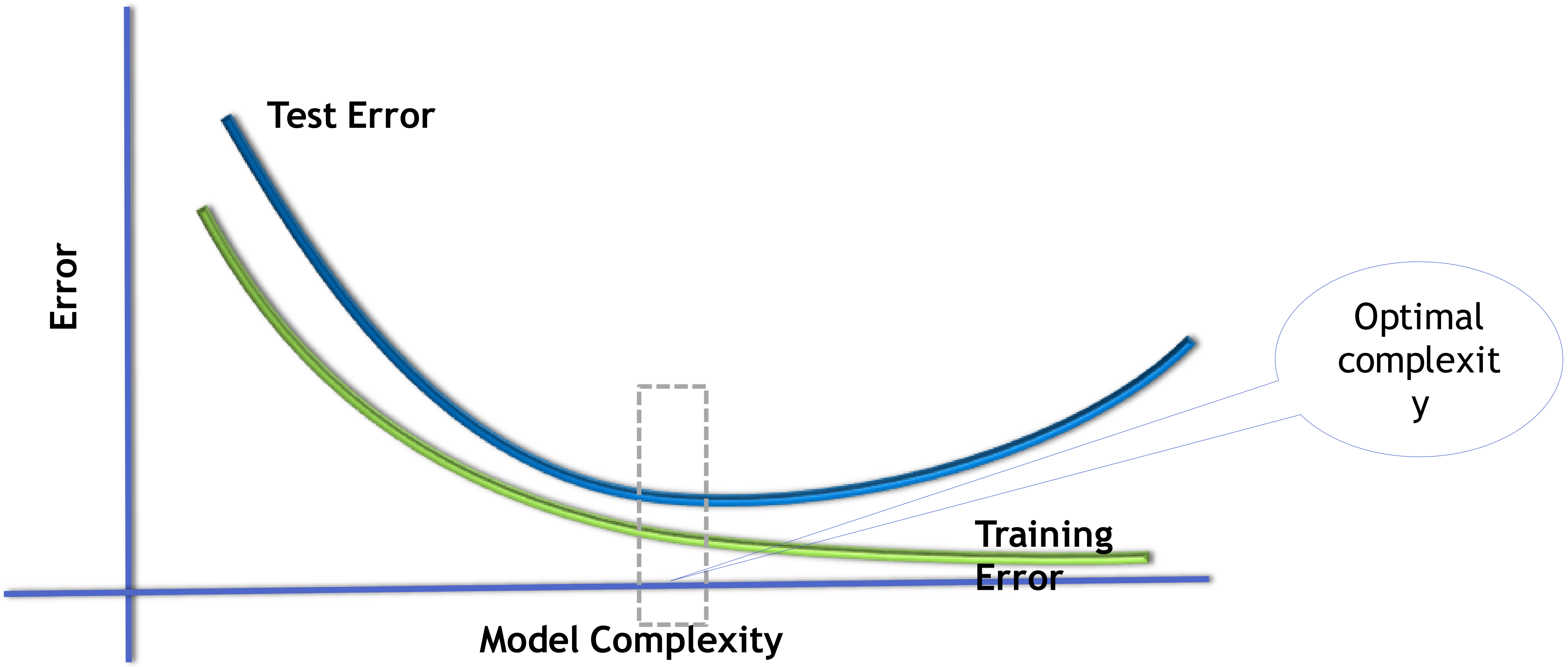
- Overall Model Squared Error = Irreducible Error + Bias^2 + Variance
- Overall error is made by bias and variance together
- High bias low variance, Low bias and high variance, both are bad for the overall accuracy of the model
- A good model need to have low bias and low variance or at least an optimal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

Choosing optimal model-Bias Variance Tradeoff

Bias Variance Tradeoff



Test and Training error



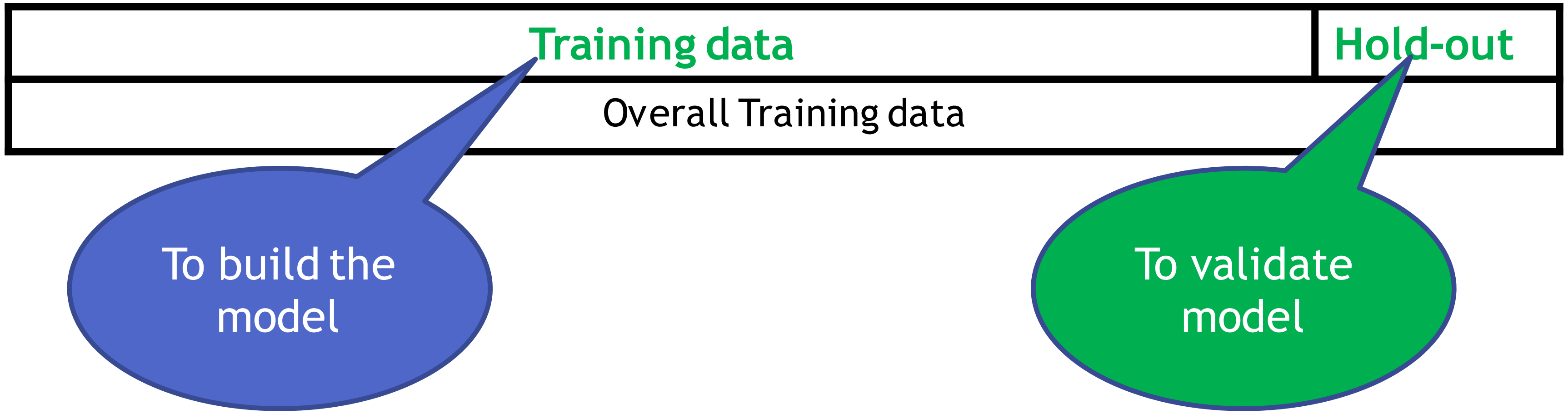
Choosing optimal model

- 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 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

Code: Holdout data Cross validation

```
tree_model = tree.DecisionTreeClassifier(max_depth=2)
tree_model.fit(X_train,y_train)

train_acc= tree_model.score(X_train,y_train)
print("train_accuracy", train_acc)

test_acc= tree_model.score(X_test,y_test)
print("test_accuracy", test_acc)
```

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 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

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?
- 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(max_depth=30)

#Simple K-Fold cross validation. 10 folds.
from sklearn.model_selection import KFold
kfold_models = KFold(n_splits=10)

from sklearn import model_selection
scores = model_selection.cross_val_score(tree_KF,X, y,cv=kfold_models)
print(scores)
print("Avg K-Fold Accuracy", scores.mean())
```

```
[0.753  0.6929 0.6022 0.7481 0.7427 0.7966 0.7678 0.653  0.8586 0.6851]
Avg K-Fold Accuracy 0.73
```

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
  - Training error is what we see and that is not the true performance metric
  - Test error plays vital role in model selection

References

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