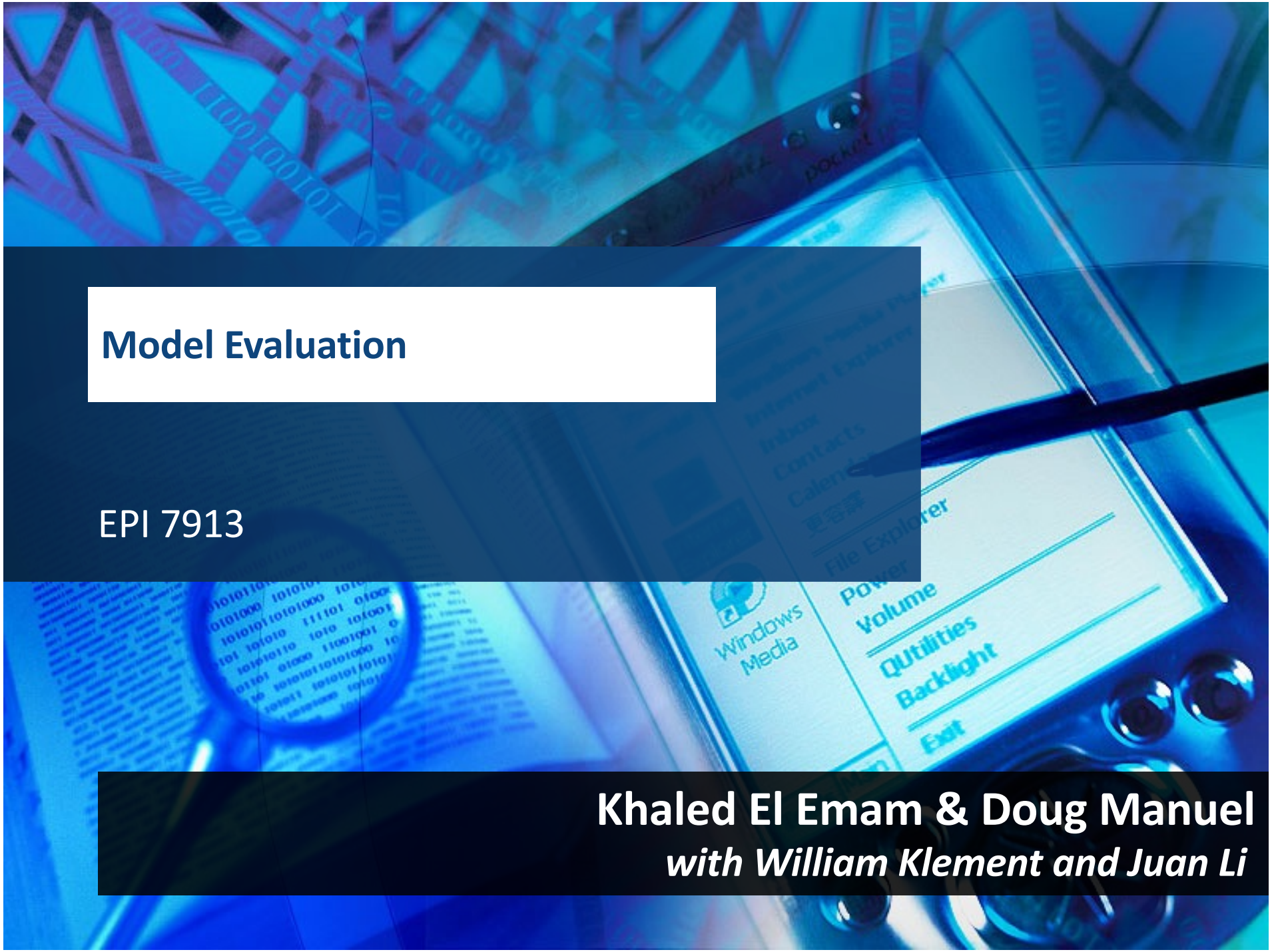


Model Evaluation

EPI 7913

Khaled El Emam & Doug Manuel
with William Klement and Juan Li

Evaluating what's being learned

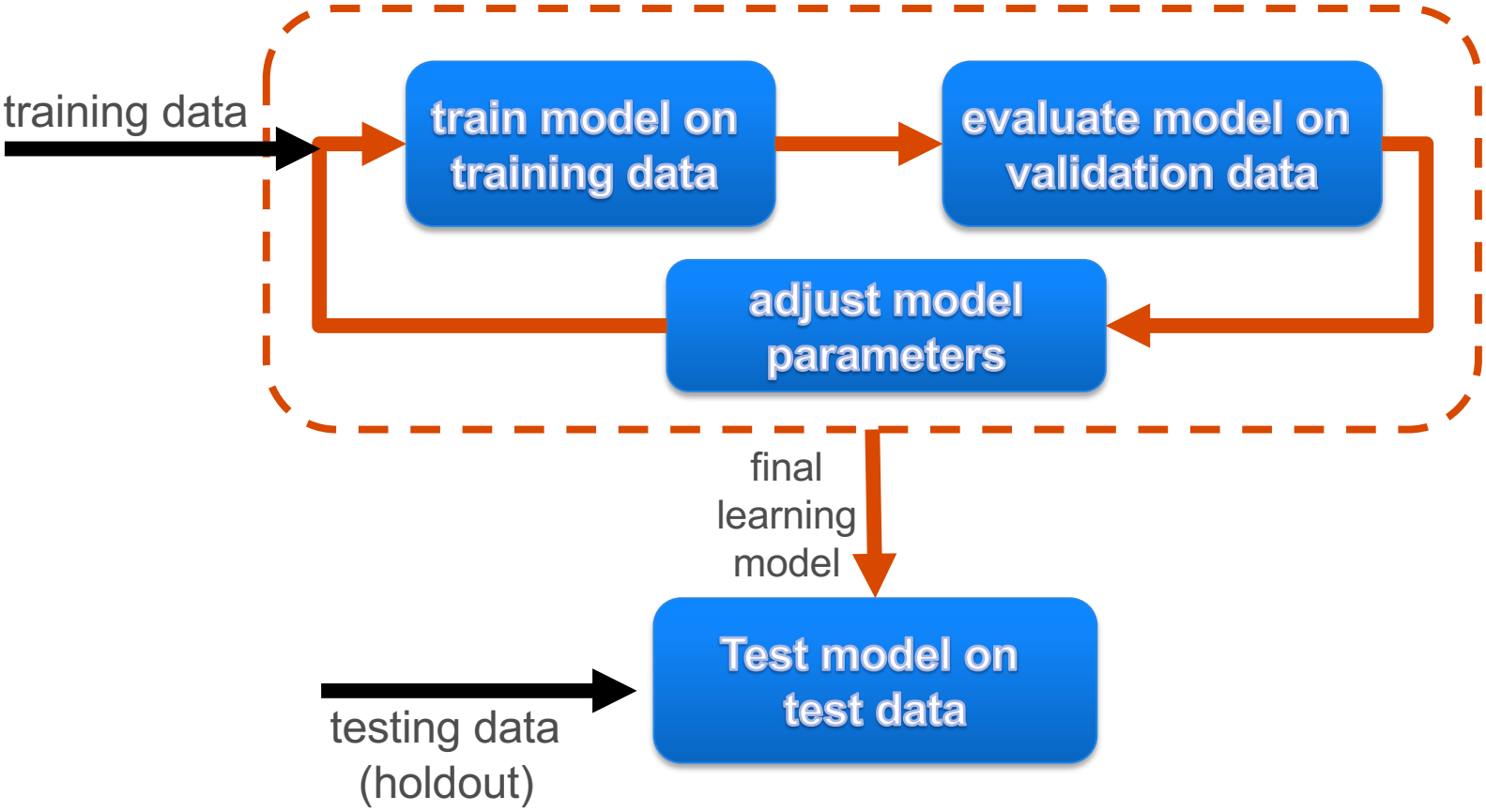
- Tree optimization – Bayesian optimization
- Model Evaluation:
  - training, testing
- N-fold cross validation:
  - leave-one-out
  - bootstrap
- Nested K-fold cross validation
- Classification Performance
  - confusion matrix
  - evaluation metrics
  - ROC curves and AUC

Model evaluation

Is the process of answering the following:

1. How well is the model performing?
2. Does the model make predictions accurate enough for deployment?
3. Will the model perform better if additional training data was made available?
4. Is the model overfitting or underfitting the data?

One round of evaluation



Random train/test strategy

1. Split the data into two stratified portions for training and testing
2. Usually, the training portion is at least twice as large as that allocated for testing
3. Stratification ensures that training and testing present similar proportions of outcome variables

Limitations:

- This does not work well if the datasets are small as observations are taken away from model training

Simple evaluation strategies

- Split data for training and testing
- Construct the model on training data
- Test model on test (holdout) data
- Using appropriate metrics, compare predictions to ground truth:
  1. Compare predictions to labels (classification)
  2. Examine the quality of estimation (regression)
- Calculate the mean and standard error for the metric used to assess model performance over repeated splits

Notebook

- Look at the notebook…

  Lecture 4 - CCHS Intro.ipynb

Splitting data by bootstrap methods

- bootstrap is a statistical method to estimate quantities about a population by averaging estimates from multiple repeated samples
- the sampling is performed with replacement
- for a dataset *D*:
  1. choose *n* = sample size
  2. while *size(S) < n:*
     - randomly select *x* observations from *D with replacement*
     - add *x* to *S*
- Generate multiple bootstrap samples.

Bootstrap error estimates

- Estimate the error for the original data where the train and test dataset are the same; this is an optimistic estimate of error and serves as a baseline: e_{orig}
- Draw say 100 bootstrap samples, and for each:
  - Build model on bootstrap data and evaluate on bootstrapped data: $e_{boot,i}$
  - build the model, and evaluate it on the full original dataset, to compute: $e_{orig,i}$
- Compute the average optimism: $O = \frac{1}{B}\sum_{i=1}^{B}\left(e_{boot,i} - e_{orig,i}\right)$
- The corrected error removes the bias from the original error estimate:

$$e_{orig} - O$$

Notebook

- Look at the notebook…

  Lecture 4 - Bootstrap on CCHS.ipynb

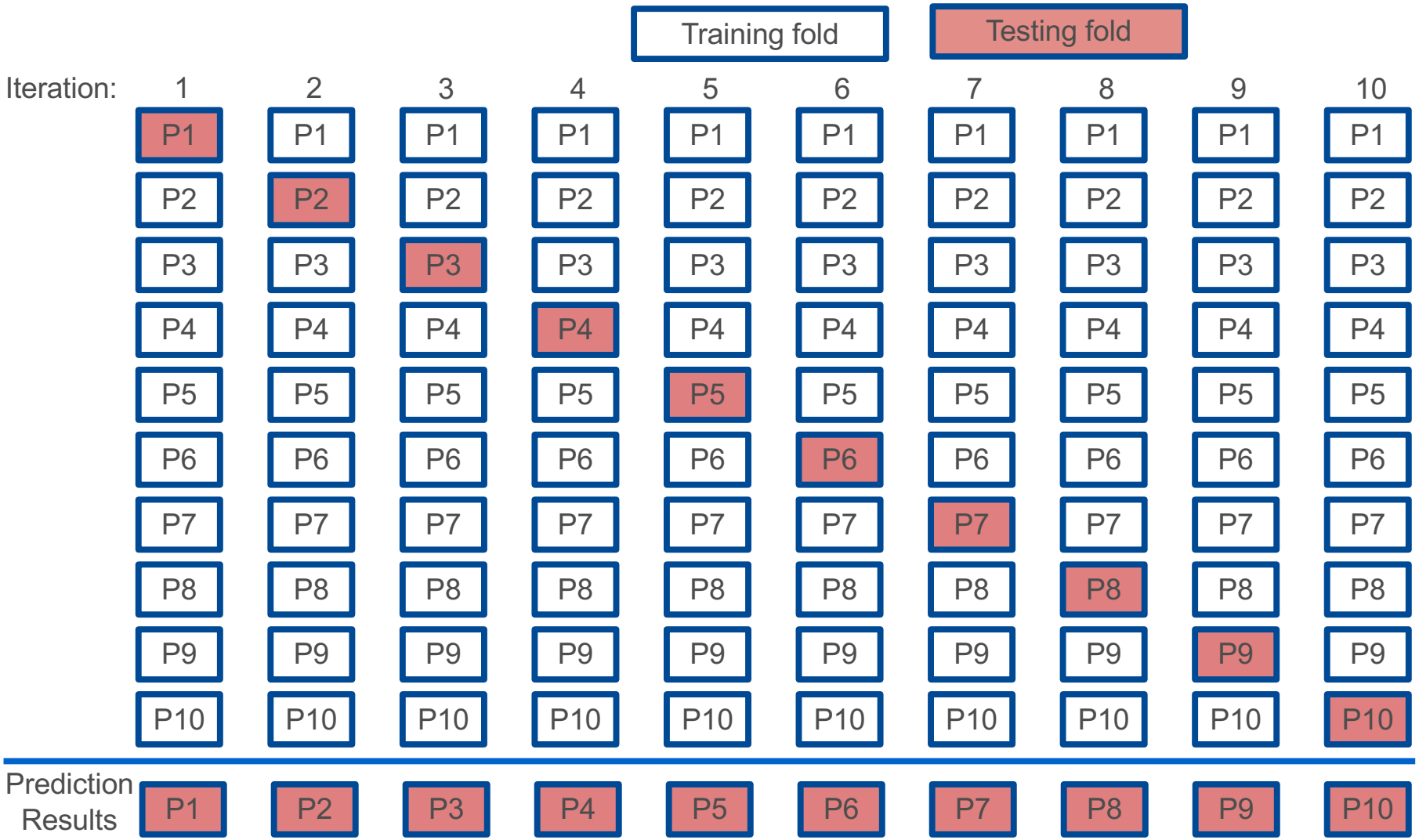
k-fold cross validation

1. split the data set into k stratified partitions

2. for i = 1 to k
   a. designate fold F_i as a holdout test fold
   b. construct a model from the remaining k-1 folds (i.e., data folds excluding F_i)
   c. Test the model on holdout data in fold F_i

3. Merge/summarize/evaluate prediction results for all test folds 1 through k

Illustration: 10-fold cross validation



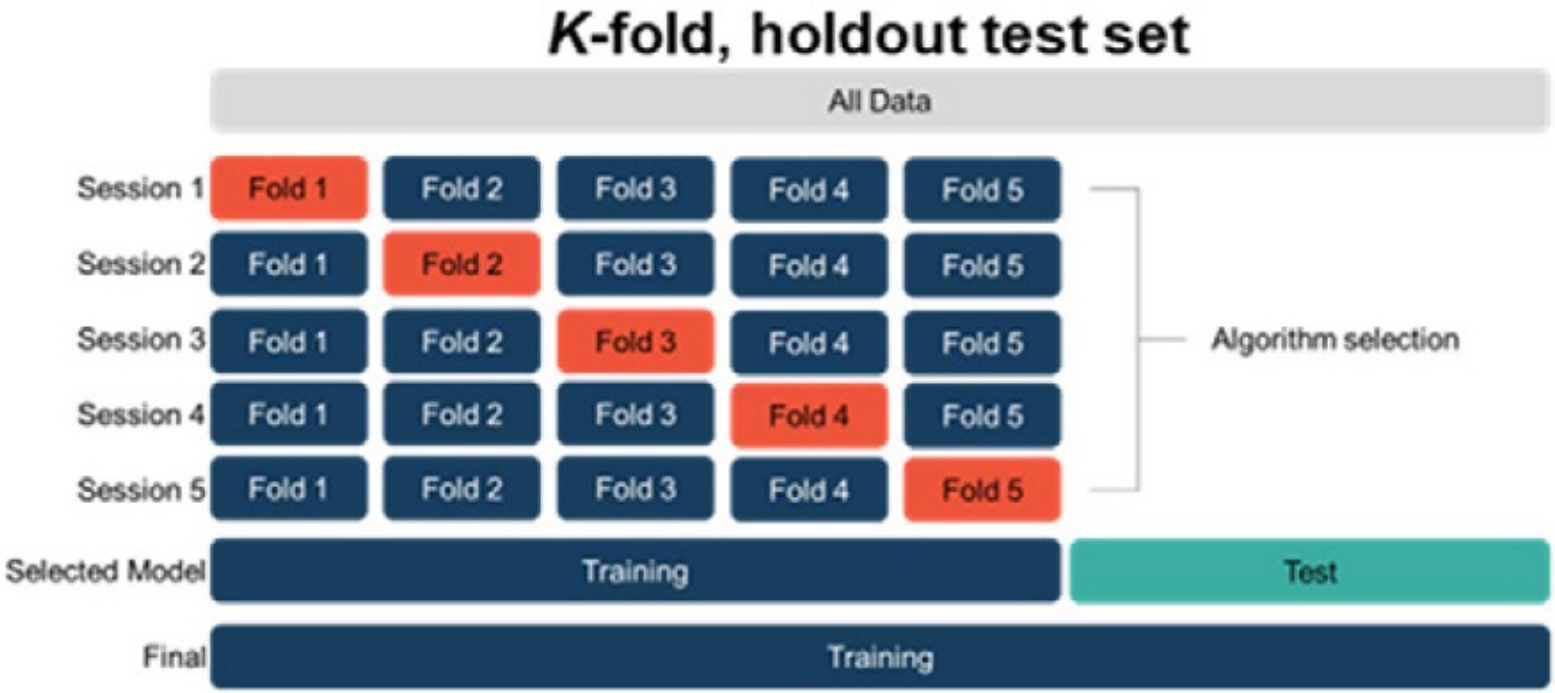
Hyperparameter tuning

- Model performance will be affected by the hyperparameters
- So as to get less biased estimates of how well the model performs, we can use a validation partition to evaluate the model performance with different hyperparameters
- We already examined train/validate/test whereby we tune the CART model hyperparameters using a grid search
- There are other optimization techniques that can be used

Bayesian optimization

- model the learning performance using a Gaussian process to help us manage the large posterior distribution
- this makes parameter tuning much more efficient and enables us to make optimal choices about what set of values and parameters to tune
- the objective is to minimize a function $f(x)$ on a limited set of data X by constructing a probabilistic model for $f(x)$
- we apply this approach to hyperparameter tuning of our CART model

https://arxiv.org/abs/1206.2944


K-fold, holdout test set
All Data
Session 1
Session 2
Session 3
Session 4
Session 5
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Algorithm selection
Selected Model
Training
Test
Final
Training

CV/test of CART

```
library(magrittr)

full_data<-epi7913A::cchs %>% dplyr::slice_sample(prop=0.1)
voutcome<-"CANHEARTbin"


# create train and test data
idx<-splitTools::partition(rep(0,nrow(full_data)), p=c(train=0.7, test=0.3),
                                        type="stratified")
train_data <- full_data[idx$train,]
test_data <- full_data[idx$test,]

# train a model with optimal hyperparameters
best_model<-sdgm::cart.bestmodel.bin(train_data, voutcome)

# predict on the test data; this is a generic predict function
preds<-predict(best_model, test_data)

# # logloss
if (!is.null(preds))
{
          test_logloss<- MLmetrics::LogLoss(preds, test_data[,voutcome] )
} else {
  test_logloss<-NA
  print("Logloss calculation failed because there are no predicted values")
}

print(paste0("Logloss on Adult Data: ", test_logloss))
```

Notebook

- See notebook:

  Lecture 4 - CART Optimization on CCHS.ipynb

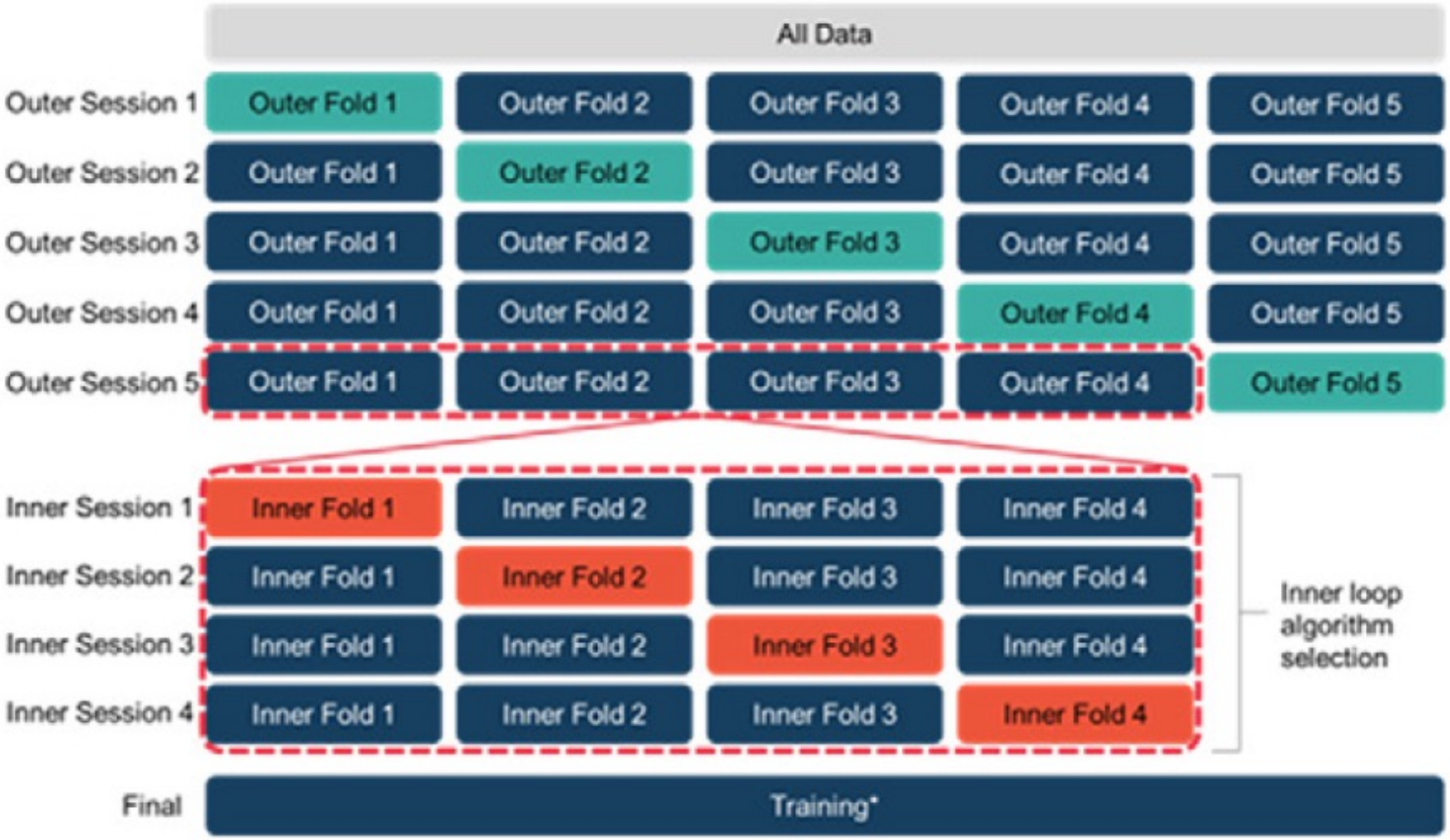
Leave-one-out cross validation

- Easy! Set K = the number of data points n
- The cross-validation strategy becomes:
  - For every data point:
    - Train on all remaining data (n -1) points
    - Test on the one holdout point

- This is a good strategy for maximizing the size of the training data
- The results may potentially be biased (depending on size and distribution of data)


Leave-one-out
All Data
Session 1
Session 2
Session 3
Session N
Final
Training

Nested CV


All Data
Outer Session 1
Outer Session 2
Outer Session 3
Outer Session 4
Outer Session 5
Outer Fold 1
Outer Fold 2
Outer Fold 3
Outer Fold 4
Outer Fold 5
Inner Session 1
Inner Session 2
Inner Session 3
Inner Session 4
Inner Fold 1
Inner Fold 2
Inner Fold 3
Inner Fold 4
Inner loop algorithm selection
Final
Training*

Nested CV of CART

```
library(magrittr)

full_data<-epi7913A::cchs %>% dplyr::slice_sample(prop=0.1)
voutcome<-"CANHEARTbin"

ll.mean<-mean(sapply(caret::createFolds(full_data[, voutcome], k=5), function(x)
{
  testInds <- x
  trnInds <- setdiff(1:nrow(full_data), testInds)

  train_data <- full_data[trnInds,]
  test_data <- full_data[testInds,]

  best_model<-sdgm::cart.bestmodel.bin(train_data, voutcome, n_iter=5)

  preds<-predict(best_model, test_data)

  if (!is.null(preds))
    test_ll<- MLmetrics::LogLoss(preds, test_data[,voutcome] )
  else
  {
    test_ll<-NA
    print("Logloss calculation failed")
  }
}))
print(ll.mean)
```

Notebook

- See notebook:

  Lecture 4 - Nested K-cross validation of CART on CCHS.ipynb

Classification performance: the confusion matrix

Actual \ Predicted	Y	N	
+	True Positives (TP)	False Negatives (FN) Type II error	Sensitivity or Recall $\text{TP rate} = \frac{TP}{TP + FN}$
-	False Positives (FP) Type I error	True Negatives (TN)	Specificity $TN\ rate = \frac{TN}{TN + FP}$
	Precision $\frac{TP}{TP + FP}$	Negative Predictive Value $\frac{TN}{TN + FN}$	Accuracy $\frac{TP + TN}{TP + TN + FP + FN}$

For a data point, comparing a predicted class to actual label can be one of four possibilities: +Y, +N, -Y, -N

Interpreting the metrics:

- **Accuracy**: proportion of correctly classified (either negative or positive)
- **Sensitivity**: proportion of correctly classified positives out of all positives
- **Specificity**: proportions of correctly classified negatives out of all negatives
- **False Positive Rate** = 1 – specificity: proportion of misclassified negatives out of all negatives
- **Precision**: proportion of correctly classified positives out of all those predicted as positive

Calculating metrics

$$accuracy = \frac{\#\ correct\ predictions}{\#\ all\ predictions}$$

$$sensitivity\ (or\ recall\ or\ TPV) = \frac{\#\ true\ positives}{\#\ all\ positives}$$

$$specificity = \frac{\#\ true\ negatives}{\#\ all\ negatives}$$

$$precision\ (PPV) = \frac{\#\ \ true\ positives}{\#\ true\ positives + \#\ false\ positives}$$

$$false\ positve\ rate\ = \frac{\#\ false\ positives}{\#\ true\ negatives + \#\ false\ positives}$$

More evaluation metrics

The F1 score is the harmonic mean of precision and recall

$$F1\ Score = \frac{2\ *precision\ *recall}{precision + recall}$$

A drawback: F1 score gives equal importance for precision and recall.

Solution: use weighted F1 score

$$F_\beta = (1 + \beta^2)\frac{precision\ *recall}{(\beta^2\ *precision) + recall}$$

$$common\ \beta = 0.5\ or\ 2$$

Calculate the confusion matrix

```
library(magrittr)

full_data <- epi7913A::cchs %>% dplyr::slice_sample(prop=0.1)
voutcome  <- "CANHEARTbin"

# create train and test data
Idx <- splitTools::partition(rep(0,nrow(full_data)), p=c(train=0.7, test=0.3),
                                        type="stratified")
train_data <- full_data[idx$train,]
test_data  <- full_data[idx$test,]

# train a model with optimal hyperparameters
best_model <- sdgm::cart.bestmodel.bin(train_data, voutcome)

# predict on the test data; this is a generic predict function
preds<-predict(best_model, test_data)

# # logloss
if (!is.null(preds))
{
          test_logloss <- MLmetrics::LogLoss(preds, test_data[,voutcome] )
} else {
  test_logloss <- NA
  print("Logloss calculation failed because there are no predicted values")
}

print(confusionMatrix(table(Pred=as.numeric(preds > 0.5),
          Actual=as.numeric(test_data[,voutcome])), positive="1"))
```

```
 Best Parameters Found:
Round = 34      minsplit = 20.0000      minbucket = 20.0000     cp = 0.0010     maxdepth = 15.0000      Value = -0.5899449
Confusion Matrix and Statistics

    Actual
Pred    0    1
   0  442  326
   1  712 1520

               Accuracy : 0.654
                 95% CI : (0.6367, 0.671)
    No Information Rate : 0.6153
    P-Value [Acc > NIR] : 6.424e-06

                  Kappa : 0.2202

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

            Sensitivity : 0.8234
            Specificity : 0.3830
         Pos Pred Value : 0.6810
         Neg Pred Value : 0.5755
             Prevalence : 0.6153
         Detection Rate : 0.5067
   Detection Prevalence : 0.7440
      Balanced Accuracy : 0.6032

       'Positive' Class : 1
```

Suppose a 2x2 table with notation

	Reference	
Predicted	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

Sensitivity = A/(A+C)

Specificity = D/(B+D)

Prevalence = (A+C)/(A+B+C+D)

*PPV = (sensitivity * prevalence)/((sensitivity*prevalence) + ((1-specificity)*(1-prevalence)))*

*NPV = (specificity * (1-prevalence))/(((1-sensitivity)*prevalence) + ((specificity)*(1-prevalence)))*

Detection Rate = A/(A+B+C+D)

Detection Prevalence = (A+B)/(A+B+C+D)

Balanced Accuracy = (sensitivity+specificity)/2

Precision = A/(A+B)

Recall = A/(A+C)

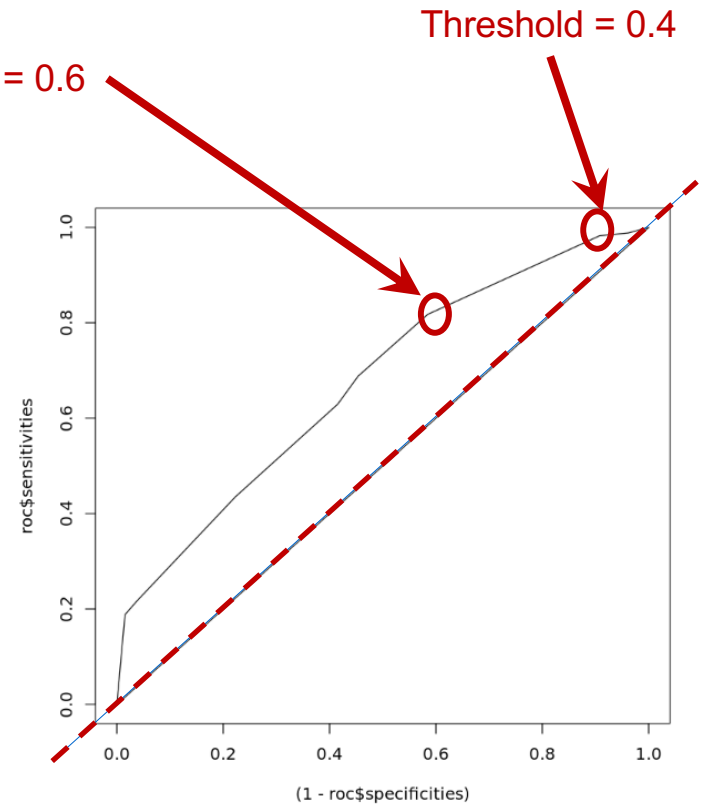
*F1 = (1+beta^2)*precision*recall/((beta^2 * precision)+recall)*

Classification decision threshold

```
# set classification decision threshold value
classificationThreshold = 0.4

# obtain classification prediction scores
test.predictions <- as.factor(ifelse(predict(best_model$model,
test_data)>classificationThreshold,1,0))

# calculate the confusion matrix
confusionMatrix(table(Pred=test.predictions,
                      Actual=test_data[,voutcome]),positive="1")


# repeat for another threshold value
classificationThreshold = 0.6
test.predictions <- as.factor(ifelse(predict(best_model$model,
                      test_data)>classificationThreshold,1,0))

confusionMatrix(table(Pred=test.predictions,
                      Actual=test_data[,voutcome]), positive="1")
```

Classification threshold

```
Confusion Matrix and Statistics

          Actual
Pred        0   1
   0       71  53
   1       59 117

               Accuracy : 0.6267
                 95% CI : (0.5692, 0.6816)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : 0.02023

                  Kappa : 0.2357

 Mcnemar's Test P-Value : 0.63660

            Sensitivity : 0.6882
            Specificity : 0.5462
         Pos Pred Value : 0.6648
         Neg Pred Value : 0.5726
             Prevalence : 0.5667
         Detection Rate : 0.3900
   Detection Prevalence : 0.5867
      Balanced Accuracy : 0.6172

       'Positive' Class : 1
```




```
Confusion Matrix and Statistics

          Actual
Pred        0   1
   0       12   3
   1      118 167

               Accuracy : 0.5967
                 95% CI : (0.5388, 0.6526)
    No Information Rate : 0.5667
    P-Value [Acc > NIR] : 0.161

                  Kappa : 0.0833

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.98235
            Specificity : 0.09231
         Pos Pred Value : 0.58596
         Neg Pred Value : 0.80000
             Prevalence : 0.56667
         Detection Rate : 0.55667
   Detection Prevalence : 0.95000
      Balanced Accuracy : 0.53733

       'Positive' Class : 1
```

Good read: https://typeset.io/pdf/an-introduction-to-roc-analysis-4n8f93uxy9.pdf

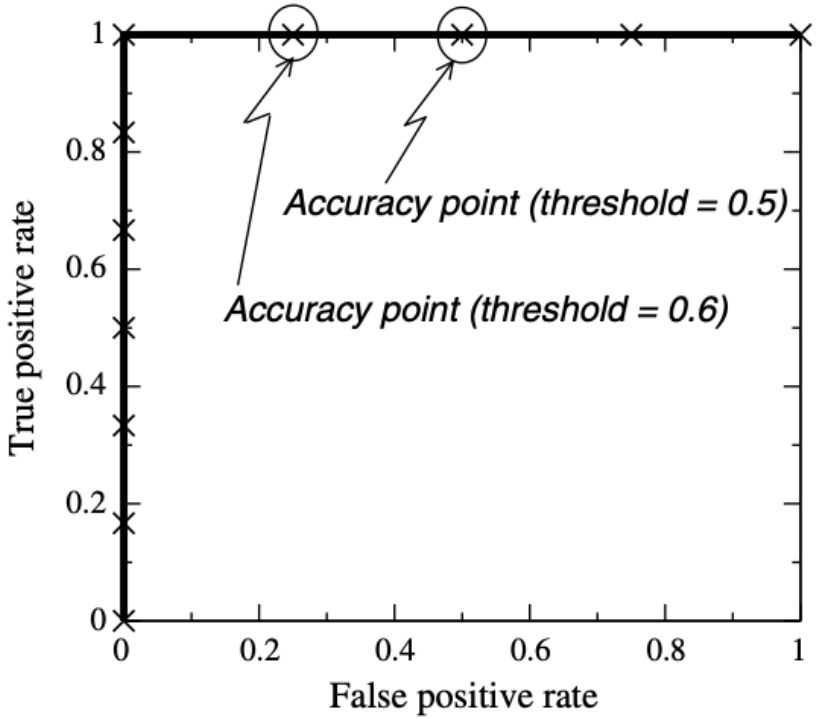
Notebook

- See notebook:

  Lecture 4 - Confusion Matrix CART on CCHS.ipynb

Receiver Operating Characteristics (ROC) curve

T. Fawcett / Pattern Recognition Letters 27 (2006) 861–874

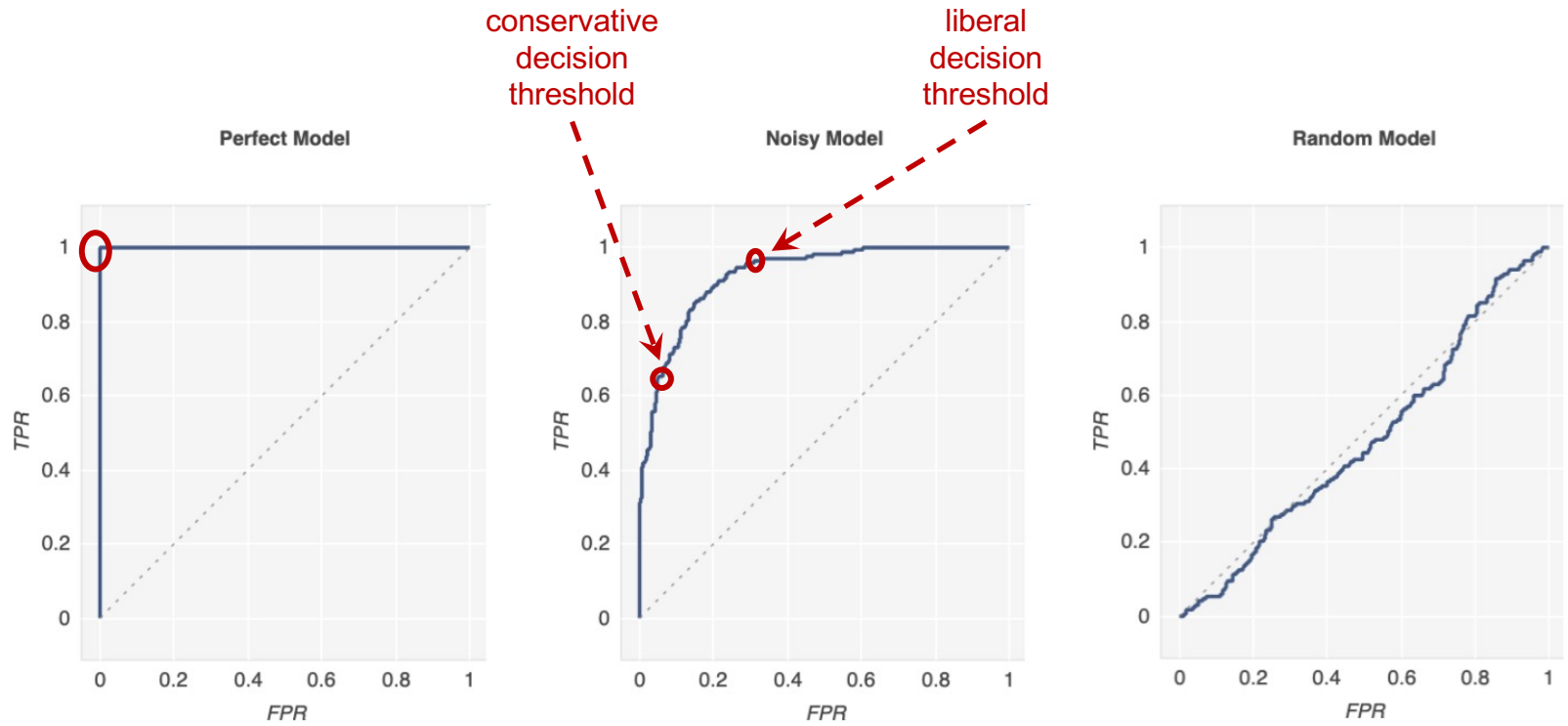


Inst no.	Class True	Class Hyp	Score
1	p	Y	0.99999
2	p	Y	0.99999
3	p	Y	0.99993
4	p	Y	0.99986
5	p	Y	0.99964
6	p	Y	0.99955
7	n	Y	0.68139
8	n	Y	0.50961
9	n	N	0.48880
10	n	N	0.44951

Fig. 4. Scores and classifications of 10 instances, and the resulting ROC curve.

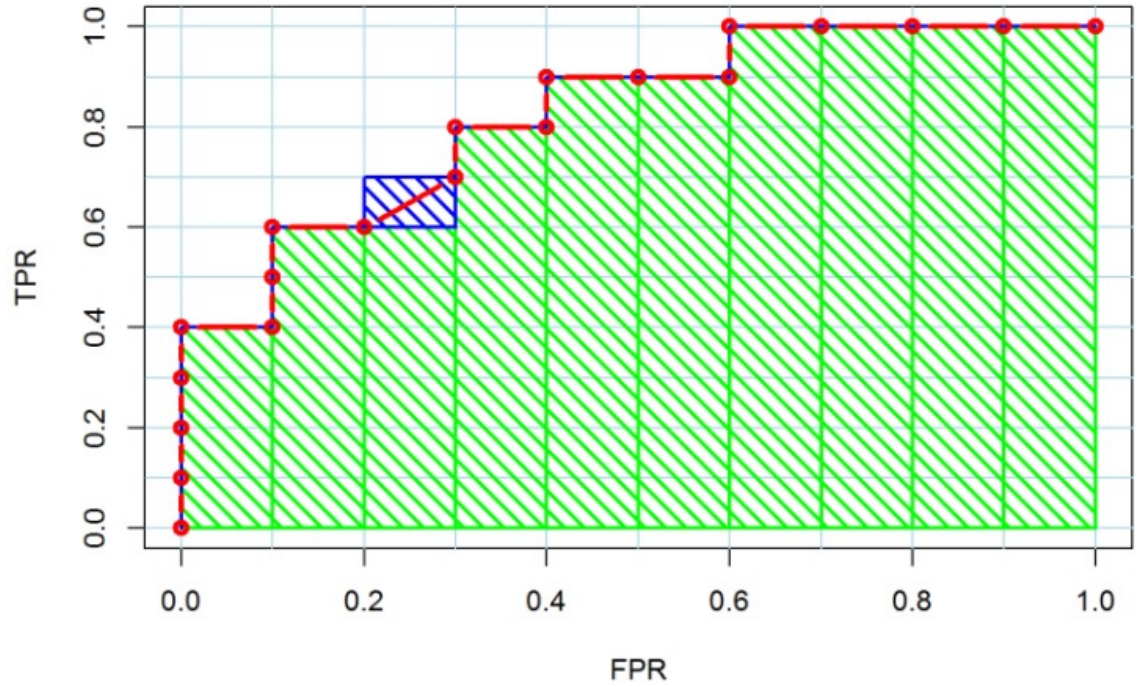
https://typeset.io/pdf/an-introduction-to-roc-analysis-4n8f93uxy9.pdf

ROC curve shapes



- a ROC curve depicts the tradeoff between TPR and FPR
- while the top left of the plot is the best solution, the dashed line represents the worst solution (a random guess)
- conservative solutions favor lower FPR but may miss some TPR
- liberal solutions favor higher TPR at the expense of FPR

Area under the ROC curve (AUC)



- Add up area in adjacent rectangles under the ROC curve
- AUC is a scalar metric independent of class ratio or classification decision thresholds
- AUC interpretation: the probability of ranking a randomly chosen positive higher than a randomly chosen negative
- However, this is an estimate and can be improved; read below..

https://blog.revolutionanalytics.com/2016/11/calculating-auc.html

Why AUC is better than accuracy?

Actual \ Predicted	Y	N	
+	True Positives (TP) (0)	False Negatives (FN) (10)	Sensitivity $\text{TP rate} = \frac{TP}{TP + FN}$
-	False Positives (FP) (0)	True Negatives (TN) (90)	Specificity $TN\ rate = \frac{TN}{TN + FP}$
	Precision $\frac{TP}{TP + FP}$	Negative Predictive Value $\frac{TN}{TN + FN}$	Accuracy $\frac{TP + TN}{TP + TN + FP + FN}$

- Imagine a model that always predicts “No” regardless of the case!
- If we test the model on data that contains 100 data points but only 10 of those are positive “+” and 90 are negative ”-”
- Populate the confusion matrix of this classifier and calculate the accuracy = $\frac{0 + 90}{10+90}$ = 90% accuracy!!!!!

For our tree example. …

```
library(magrittr)

full_data <- epi7913A::cchs %>% dplyr::slice_sample(prop=0.1)
voutcome  <- "CANHEARTbin"

# create train and test data
idx<-splitTools::partition(rep(0,nrow(full_data)), p=c(train=0.7, test=0.3),
                                                   type="stratified")
train_data <- full_data[idx$train,]
test_data <- full_data[idx$test,]

# train a model with optimal hyperparameters
best_model<-sdgm::cart.bestmodel.bin(train_data, voutcome)

# predict on the test data; this is a generic predict function
preds<-predict(best_model, test_data)

# # AUC
if (!is.null(preds))
{
  test_auc<- sdgm::auc(preds, test_data[,voutcome] )
} else {
  test_auc<--NA
  print("AUC calculation failed because there are no predicted values")
}
print(paste0("AUC on CCHS Data: ", test_auc))
```

The R package “measures” is also useful, please do look it up:

https://cran.r-project.org/web/packages/measures/index.html

Notebook

- See notebook:

  Lecture 4 - AUC on CCHS.ipynb

Brier score

- The Brier score measures the accuracy of probabilistic predictions assigned to mutually exclusive discrete outcomes.

$$\frac{1}{n} * \sum_{i=1}^{n} (p_i - oi)^2$$

$where$:

- p_i $is\ the\ predicted\ probability$
- o_i $is\ the\ observed\ value\ \in \{0,1\}$

```
# for our example
sdgm::brier(test.predictions[,2], test$CANHEARTbin)
```

For our tree example. …

```
library(magrittr)

full_data <- epi7913A::cchs %>% dplyr::slice_sample(prop=0.1)
voutcome  <- "CANHEARTbin"

# create train and test data
idx<-splitTools::partition(rep(0,nrow(full_data)), p=c(train=0.7, test=0.3),
                                                   type="stratified")
train_data <- full_data[idx$train,]
test_data <- full_data[idx$test,]

# train a model with optimal hyperparameters
best_model <- sdgm::cart.bestmodel.bin(train_data, voutcome)

# predict on the test data; this is a generic predict function
preds<-predict(best_model, test_data)

# # brier
if (!is.null(preds))
{
  test_brier<- sdgm::brier(preds, test_data[,voutcome] )
} else {
  test_brier<-NA
  print("Brier calculation failed because there are no predicted values")
}
print(paste0("Brier Score on Adult Data: ", test_brier))
```

The R package “measures” is also useful, please do look it up:

https://cran.r-project.org/web/packages/measures/index.html

Notebook

- See notebook:

  Lecture 4 - Brier on CCHS.ipynb