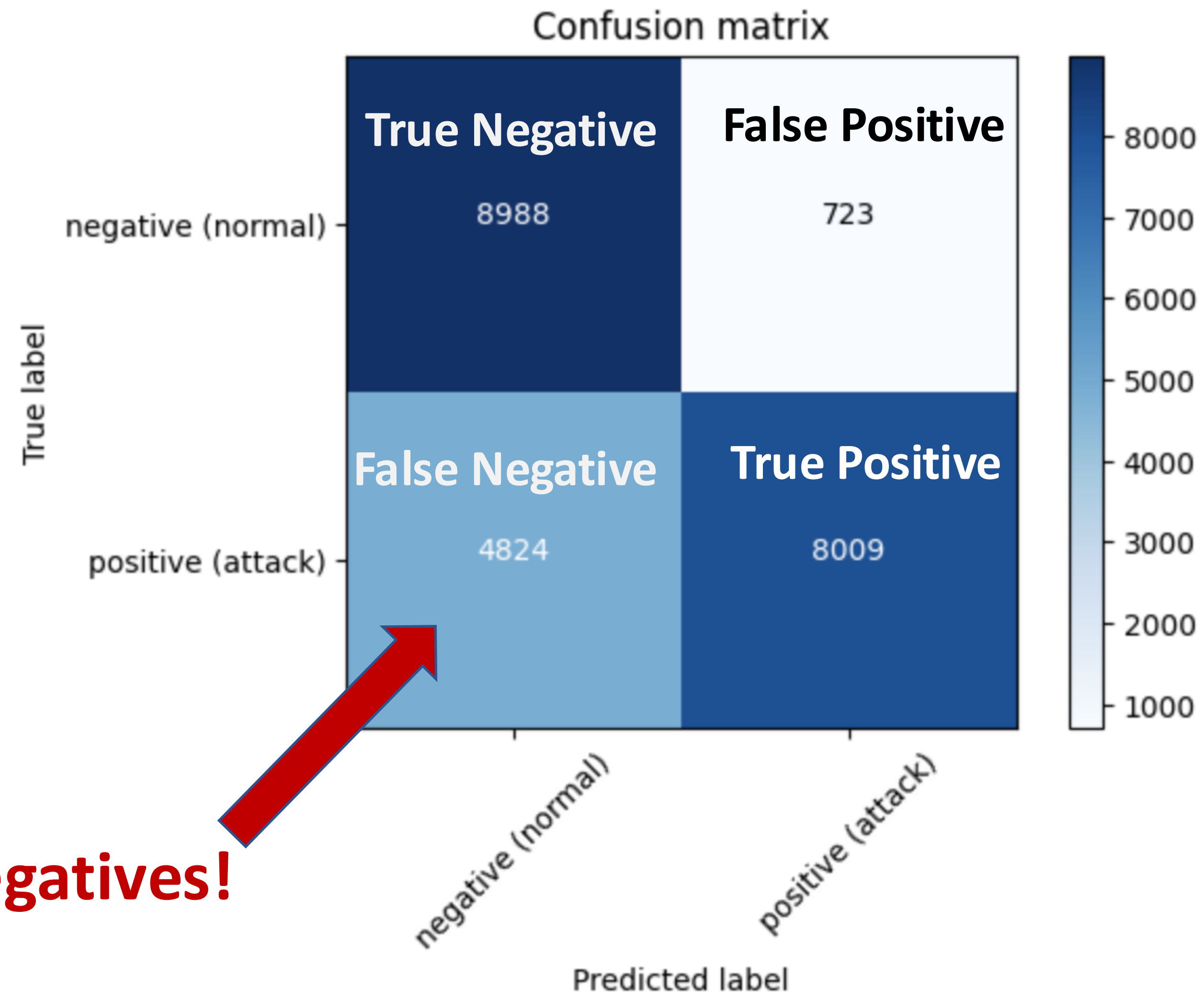
Machine Learning in Spark: Evaluation and Hyper-Parameter Tuning

Lecture 9 for 14-763/18-763

Guannan Qu

Sept 30, 2024

Evaluation: Confusion Matrix



Too many false negatives!

Evaluation: Confusion Matrix

Thresholding

If prob. > threshold, then predict ”attack”.
Otherwise predict “normal”.
Default threshold is 0.5

Input → Score → Probability → Output

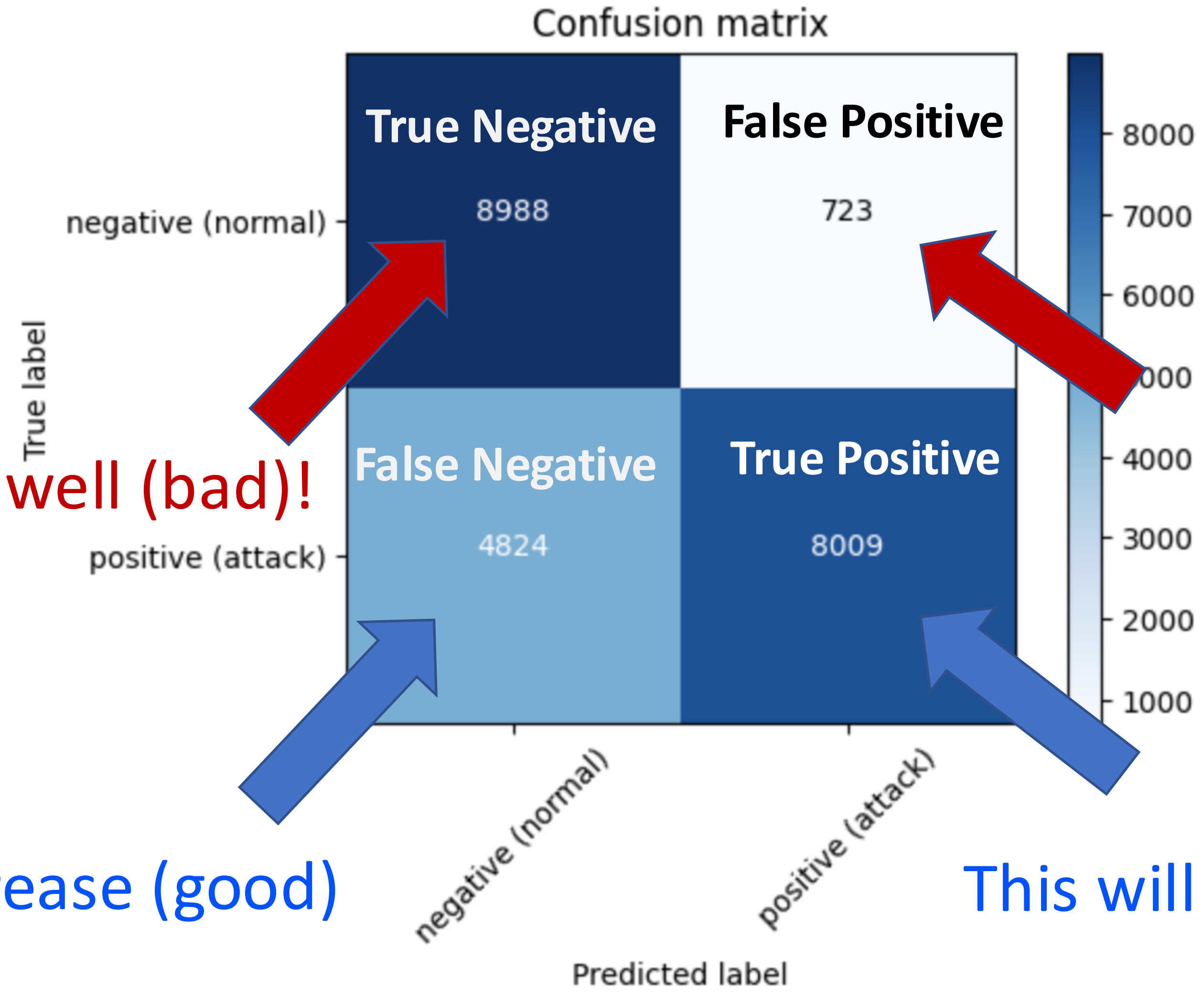
What if we adjust the threshold?

Evaluation: Confusion Matrix

Thresholding

If prob. > threshold, then predict ”attack”.
Otherwise predict “normal”.

Decrease the threshold -> more records will be predicted positive
-> less records will be predicted negative



increase as well (bad)!

decrease as well (bad)!

This will decrease (good)

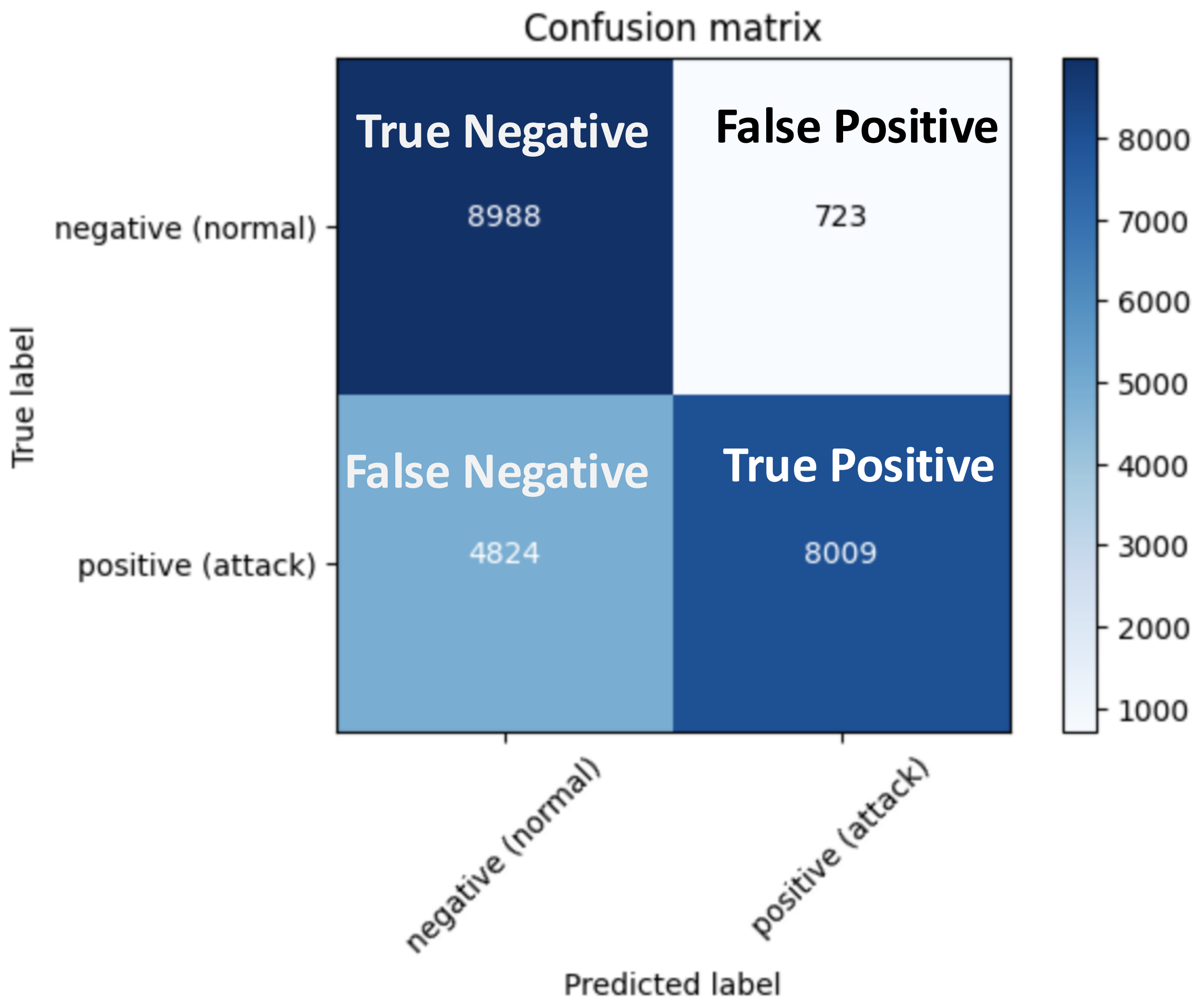
This will increase (good)

Decrease the threshold -> more records will be predicted positive

-> less records will be predicted negative

How do we strike the balance?

ROC (Receiver Operating Characteristic) curve will help!



False Positive Rate

(the smaller the better)

$$\frac{FP}{FP + TN}$$

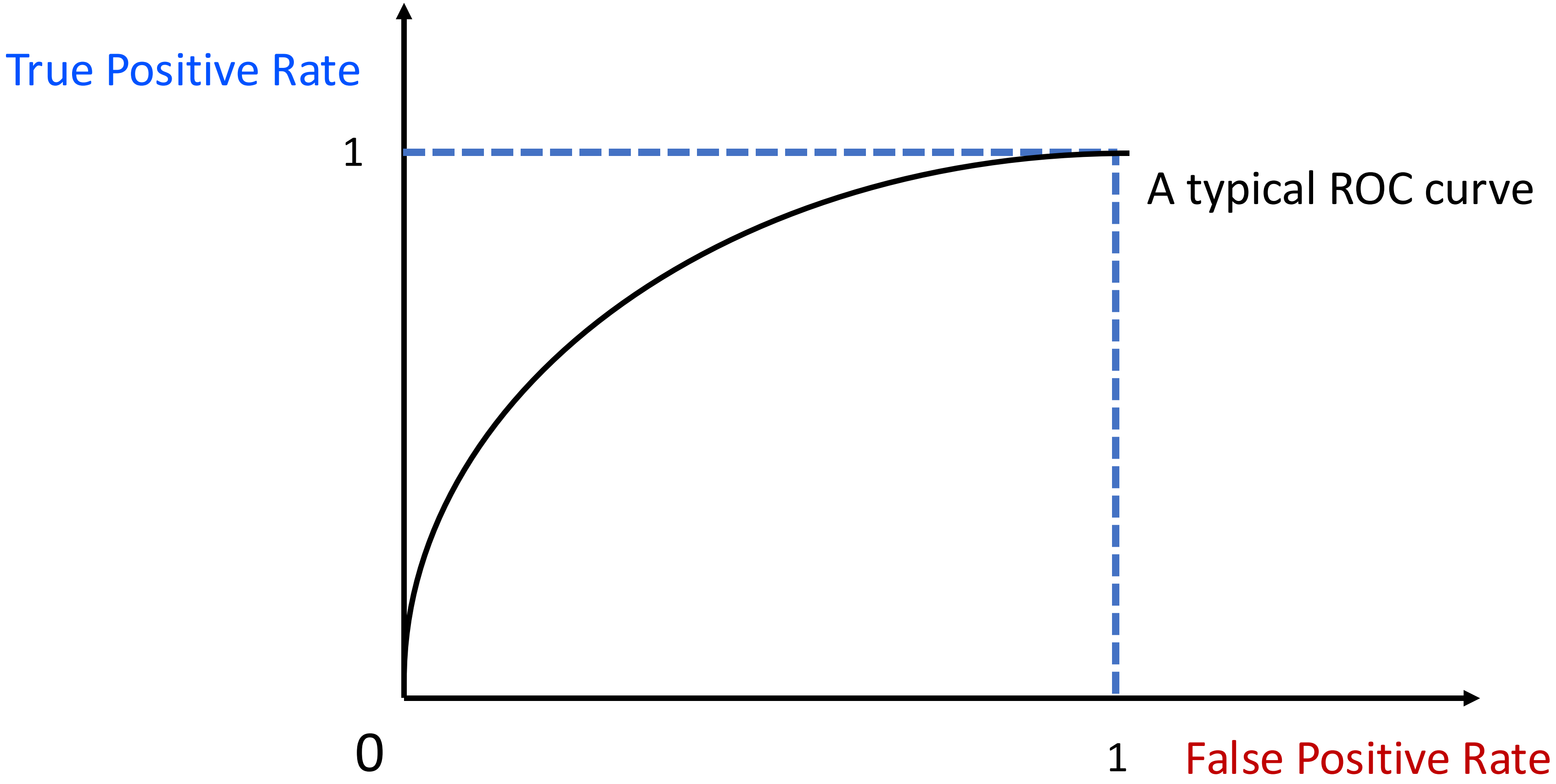
True Positive Rate

(the larger the better)

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

ROC curve plots the True Positive Rate against False Positive Rate when adjusting the threshold from 0 to 1!

Evaluation: ROC Curve



ROC curve plots the True Positive Rate against False Positive Rate when adjusting the threshold from 0 to 1!

```python
pred_prob = predictions.select("probability")
to_array = F.udf(lambda v: v.toArray().tolist(), T.ArrayType(T.FloatType()))
pred_prob = pred_prob.withColumn('probability', to_array('probability'))
pred_prob = pred_prob.toPandas()
pred_prob_nparray = np.array(pred_prob['probability'].values.tolist())

fpr, tpr, thresholds = roc_curve(outcome_true, pred_prob_nparray[:,1])

# plot the roc curve for the model
plt.plot(fpr, tpr, linestyle='--', label='Logistic')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

Vector of false positive rates, true positive rates when varying thresholds

Column vector of probability of attack

```
pred_prob = predictions.select("probability")
to_array = F.udf(lambda v: v.toArray().tolist(), T.ArrayType(T.FloatType()))
pred_prob = pred_prob.withColumn('probability', to_array('probability'))
pred_prob = pred_prob.toPandas()
pred_prob_nparray = np.array(pred_prob['probability'].values.tolist())

fpr, tpr, thresholds = roc_curve(outcome_true, pred_prob_nparray[:,1])

# plot the roc curve for the model
plt.plot(fpr, tpr, linestyle='--', label='Logistic')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

Column vector of true label

Values of the thresholds that was used to calculate fpr, tpr

This part gets the probability and convert it to nparray!

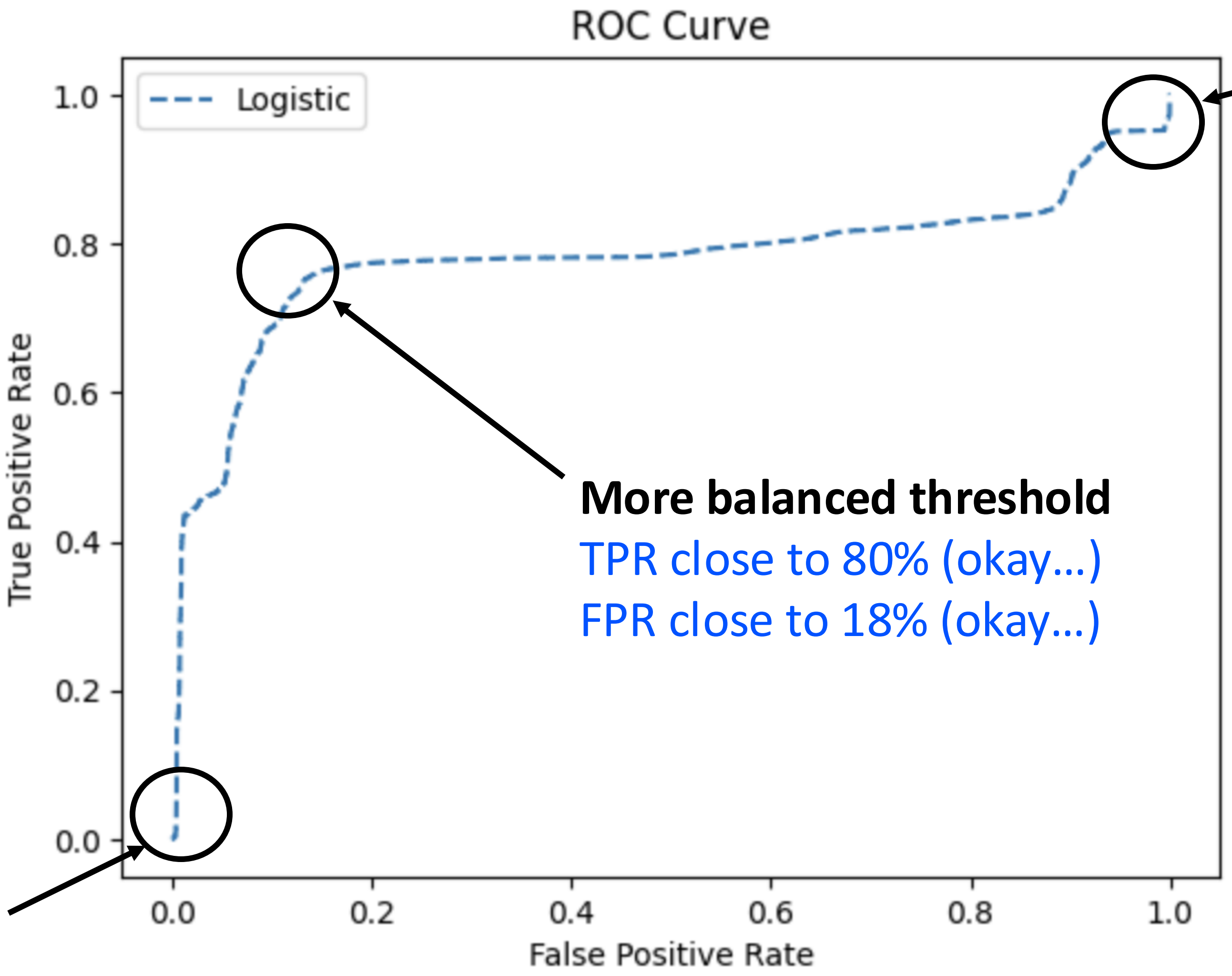
```
pred_prob = predictions.select("probability")
to_array = F.udf(lambda v: v.toArray().tolist(), T.ArrayType(T.FloatType()))
pred_prob = pred_prob.withColumn('probability', to_array('probability'))
pred_prob = pred_prob.toPandas()
pred_prob_nparray = np.array(pred_prob['probability'].values.tolist())
```

```
fpr, tpr, thresholds = roc_curve(outcome_true, pred_prob_nparray[:,1])
```

```
# plot the roc curve for the model
plt.plot(fpr, tpr, linestyle='--', label='Logistic')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

This part does the plotting!

Evaluation: ROC Curve

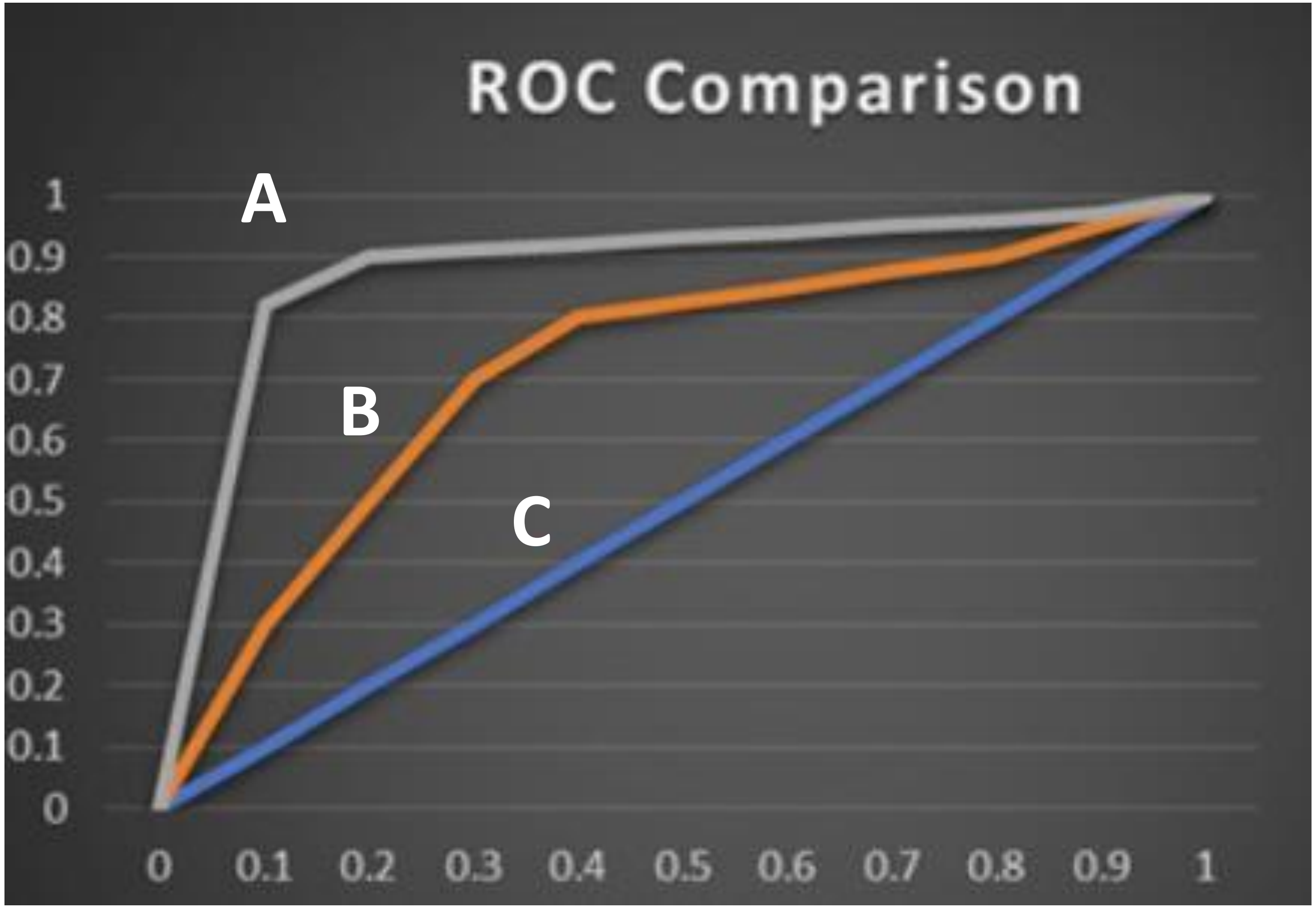


Threshold close to 0
TPR close to 1 (GOOD)
FPR close to 1 (BAD)

More balanced threshold
TPR close to 80% (okay…)
FPR close to 18% (okay…)

Threshold close to 1
TPR close to 0 (BAD)
FPR close to 0 (GOOD)

Evaluation: ROC Curve

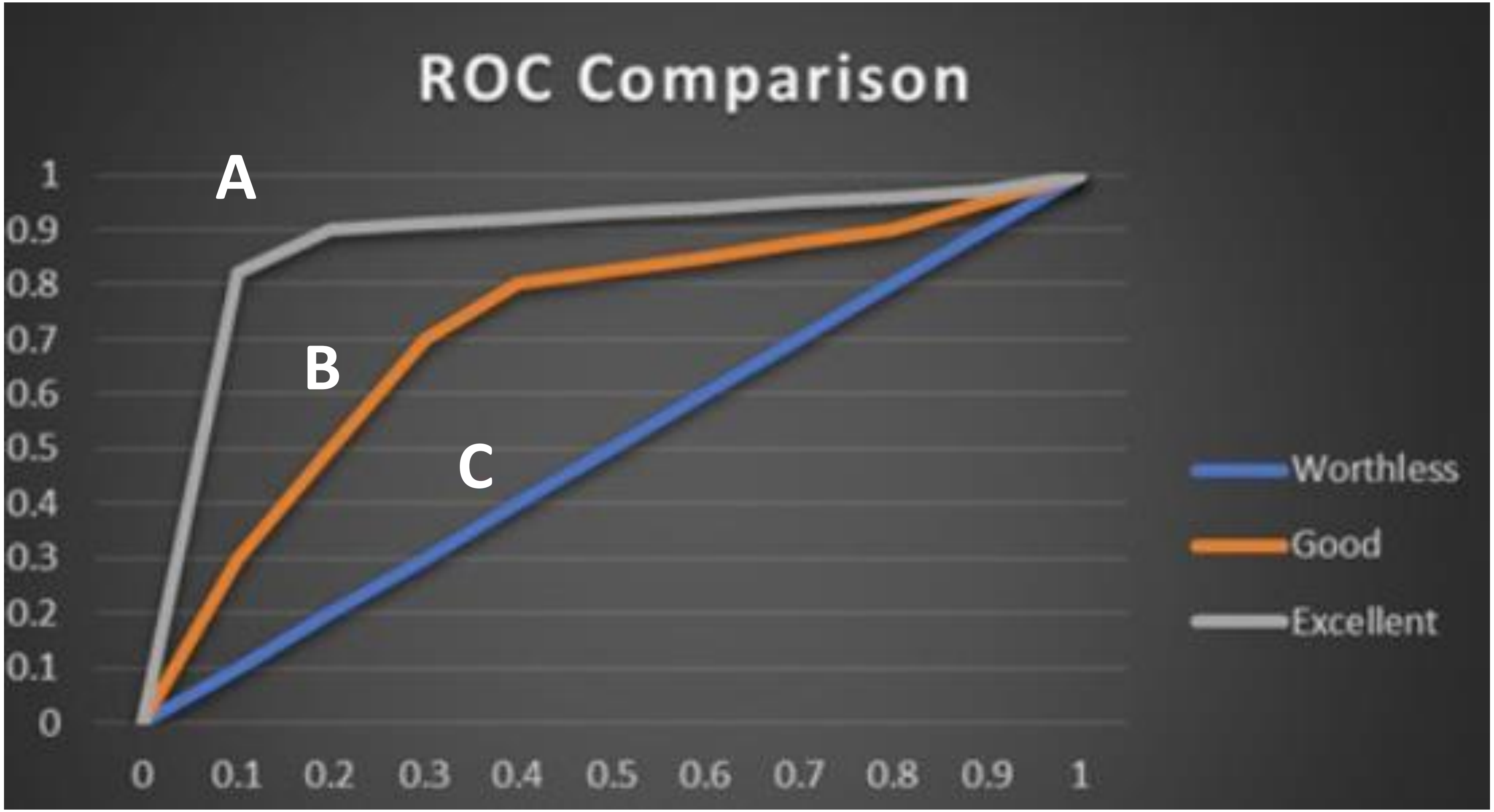


True Positive

False Positive

Which one of the three is the best?

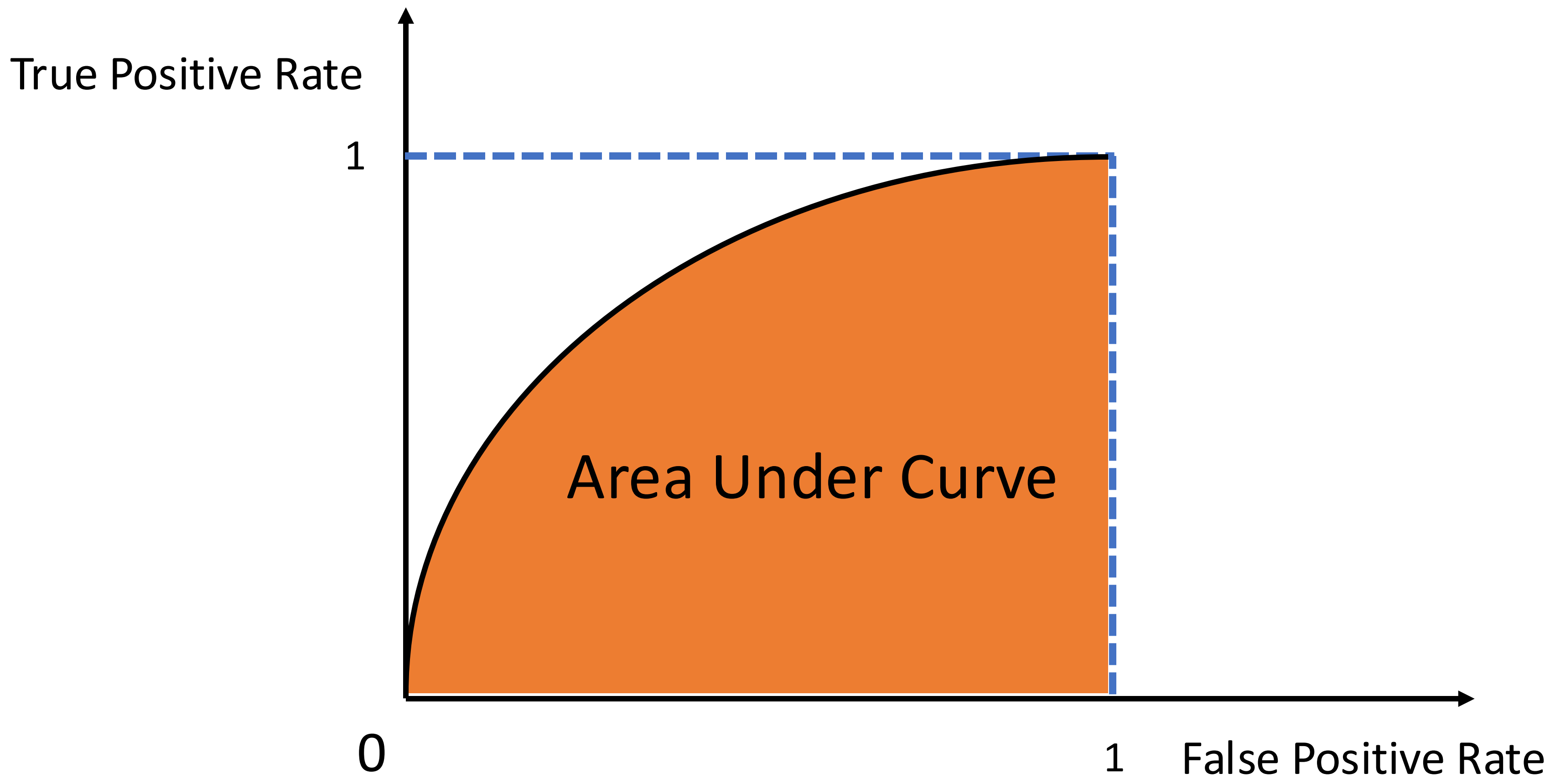
Evaluation: ROC Curve



True Positive

False Positive

Evaluation: Area Under Curve (AUC)



Evaluation: Area Under Curve (AUC)

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction',
    labelCol='outcome', metricName='areaUnderROC')
print("Area under the curve is: ", evaluator.evaluate(predictions))
```

```
Area under the curve is:  0.7795687241590551
```

How can we improve?

Summary for today

Phase II: ML Modeling

Identify the Proper ML Model

↓

Data Engineering & Preprocess

↓

Train, Evaluate, and Parameter Tuning

↓

Obtain Final Tuned Model

Train

Evaluation

- Accuracy
- Confusion Matrix
- ROC/AUC

Next: Parameter Tuning via Cross Validation

Hyper-Parameters

We didn't specify any hyper-parameters

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol = 'features', labelCol = 'outcome')

lrModel = lr.fit(nslkdd_df) # fit the logistic regression model to the training dataset
```

LogisticRegression ¶

class pyspark.ml.classification.**LogisticRegression**(**, featuresCol: str = 'features', labelCol: str = 'label', predictionCol: str = 'prediction', maxIter: int = 100, regParam: float = 0.0, elasticNetParam: float = 0.0, tol: float = 1e-06, fitIntercept: bool = True, threshold: float = 0.5, thresholds: Optional[List[float]] = None, probabilityCol: str = 'probability', rawPredictionCol: str = 'rawPrediction', standardization: bool = True, weightCol: Optional[str] = None, aggregationDepth: int = 2, family: str = 'auto', lowerBoundsOnCoefficients: Optional[pyspark.ml.linalg.Matrix] = None, upperBoundsOnCoefficients: Optional[pyspark.ml.linalg.Matrix] = None, lowerBoundsOnIntercepts:*

Hyper-Parameters

- maxIter: maximum number of iterations
  - positive integer values

- regParam: regularization parameter
  - nonnegative real numbers, default = 0

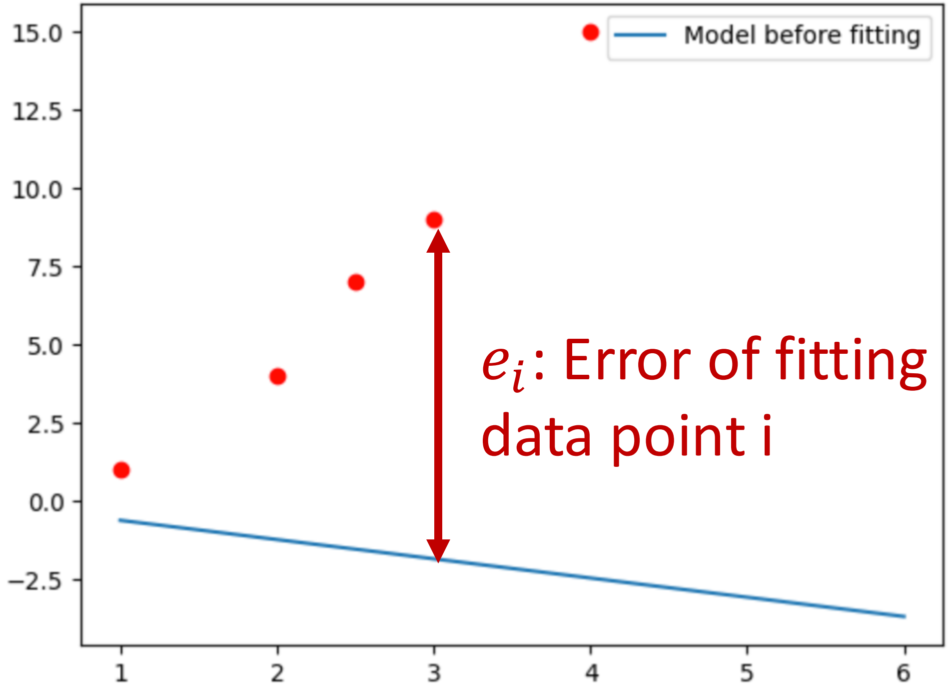
What do these parameters mean?

How does training work?

Linear Model: $y = ax + b$

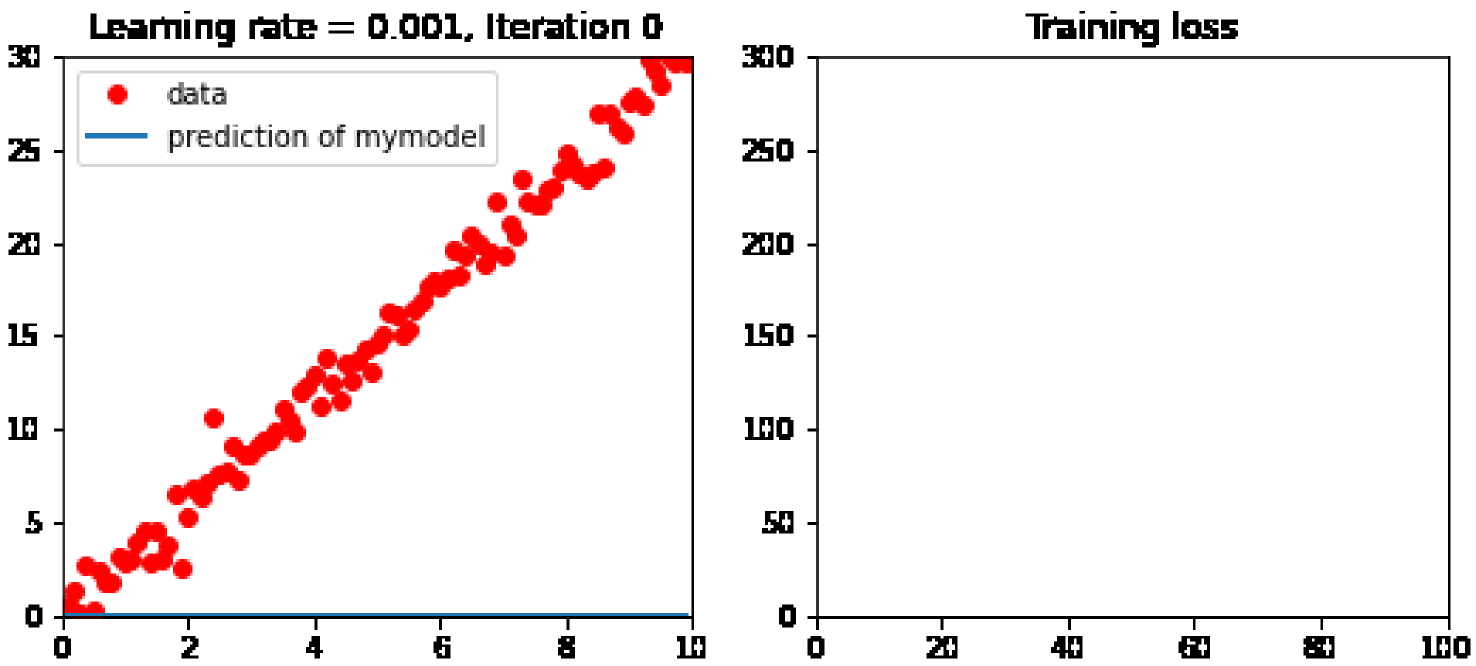
Model Parameters: a, b

$$loss(a, b) = \frac{1}{N}\sum_i \underbrace{\left(y_i - (ax_i + b)\right)}_{e_i}{}^2$$



The training/fitting process finds the a,b with the smallest loss!

How does training work?



hyper-parameters affect the outcome of training

maxIter: maximum iterations to run
regParam: regularization parameter in the loss - > can help reduce over fitting

How does fitting really work? (Logistic Regression)

Linear Function

$Score = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$

Logistic Function

$$Prob. = \frac{1}{1 + e^{-Score}}$$

Thresholding

Predict attack if prob>0.5

Input → Score → Probability → Output

Input: Feature vector $(x_1, \ldots, x_n)$

Score: The higher the score the larger the odds of "attack"

Probability: Probability of "attack"

Output: Attack or normal

How does fitting really work? (Logistic Regression)

Linear Function

$Score = w_1x_1 + w_2x_2 + b$

Logistic Function

$Prob. = \dfrac{1}{1 + e^{-Score}}$

Thresholding

Predict attack if prob>0.5

Input → Score → Probability → Output

Input: Feature vector (x_1, x_2)

Score: The higher the score the larger the odds of "attack"

Probability: Probability of "attack"

Output: Attack or normal

How does fitting really work? (Logistic Regression)

Linear Function

$$Score = w_1x_1 + w_2x_2 + b$$

Logistic Function

$$Prob. = \frac{1}{1 + e^{-Score}}$$

Thresholding

Predict attack if prob>0.5

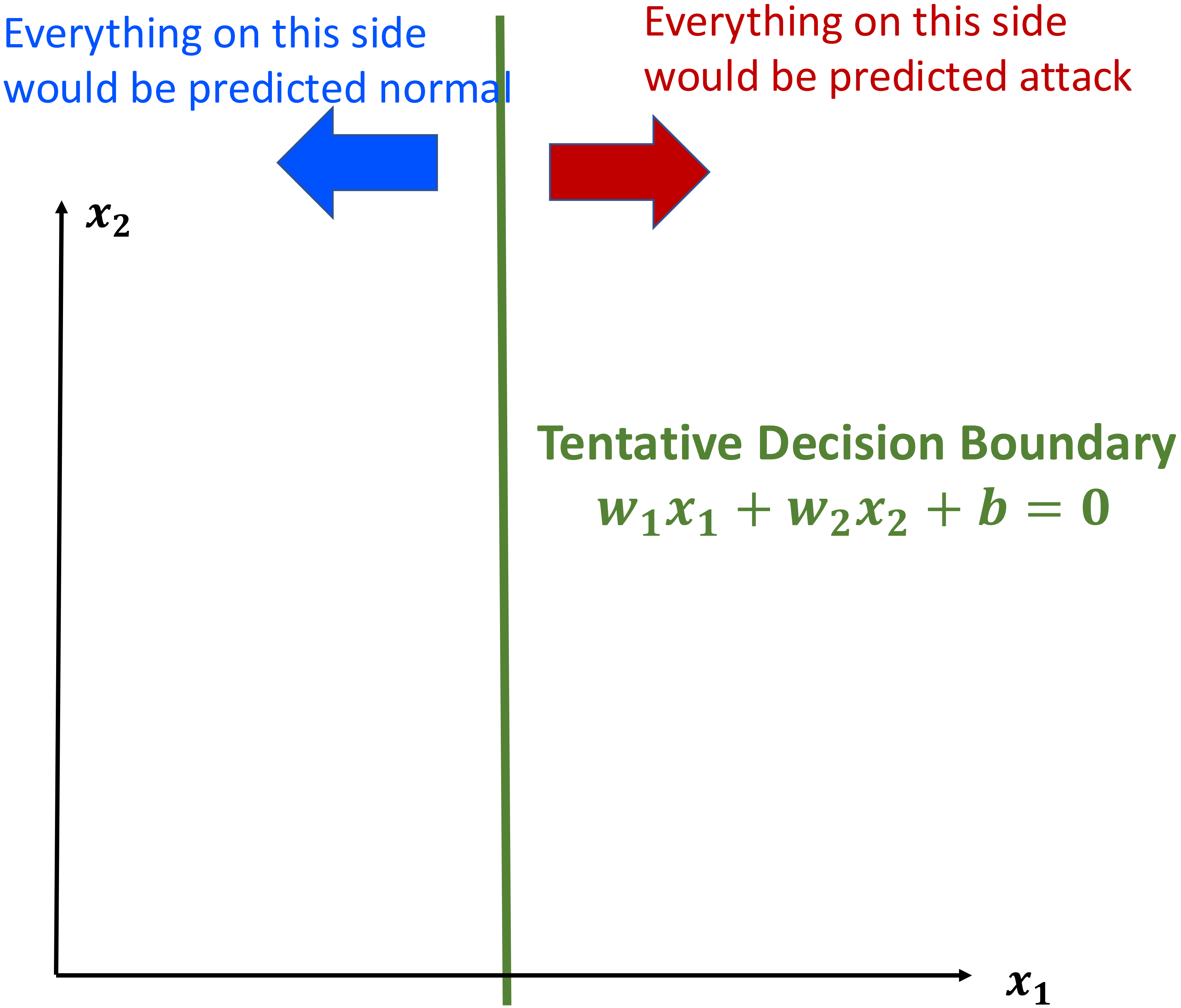
$$\text{Input} \longrightarrow \text{Score} \longrightarrow \text{Probability} \longrightarrow \text{Output}$$

Cut-off score: 0

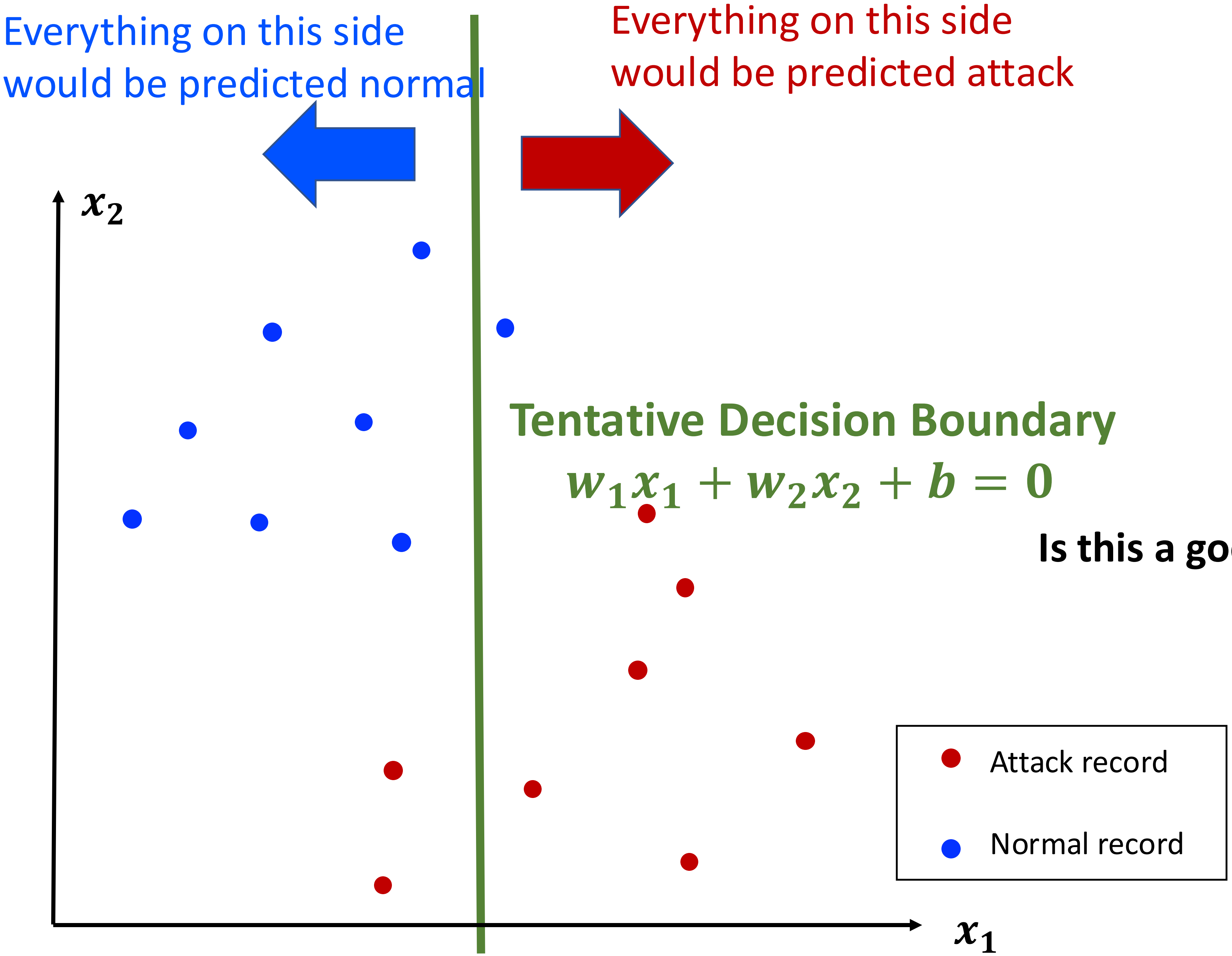
Cut-off probability: 0.5

Equivalent way to represent the decision rule of logistic regression

If $w_1x_1 + w_2x_2 + b > 0$, predict attack
If $w_1x_1 + w_2x_2 + b \leq 0$, predict normal



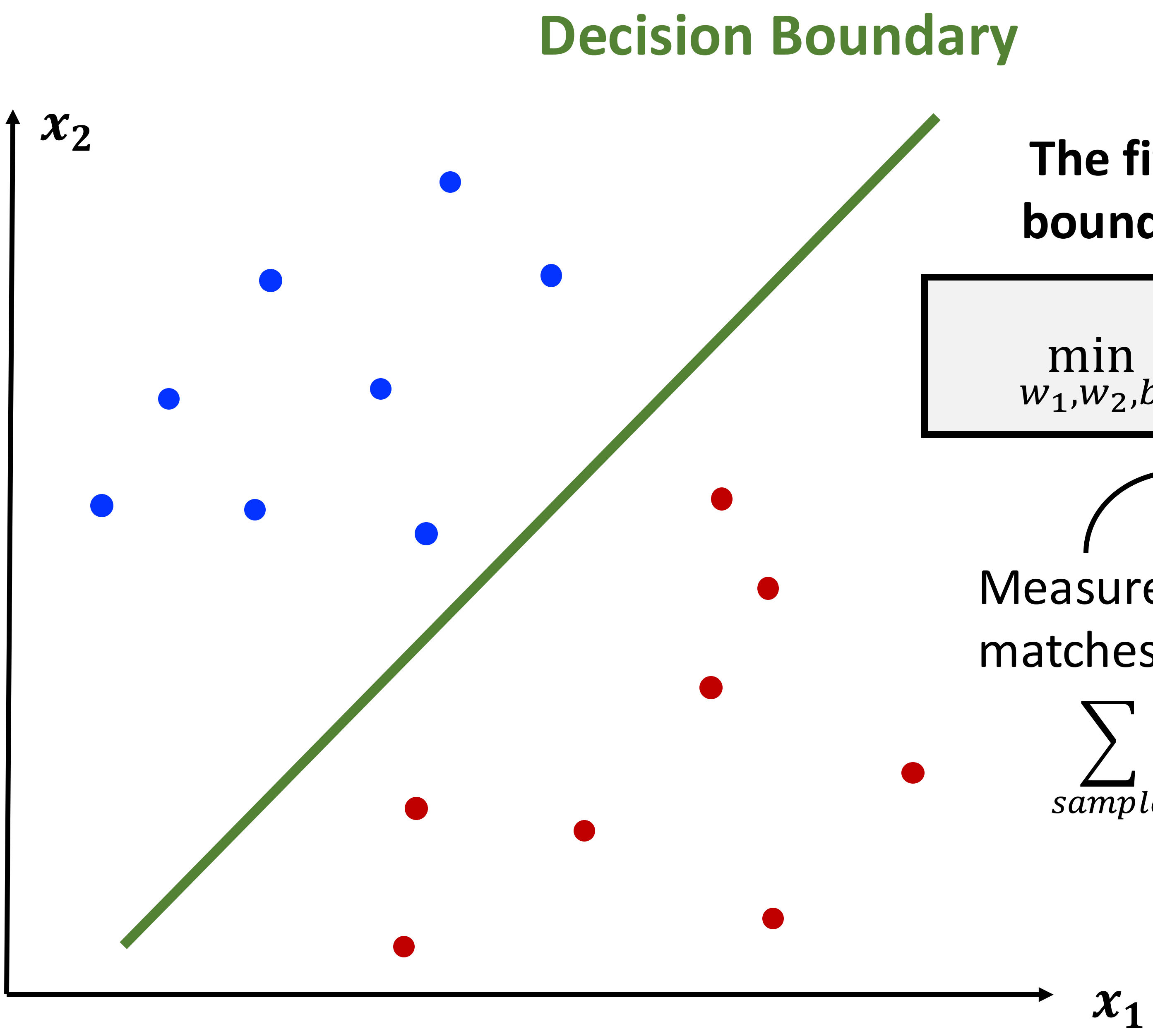
Everything on this side
would be predicted normal
Everything on this side
would be predicted attack
x_2
Tentative Decision Boundary
$w_1x_1 + w_2x_2 + b = 0$
x_1



Tentative Decision Boundary

$$w_1 x_1 + w_2 x_2 + b = 0$$

Is this a good decision boundary?

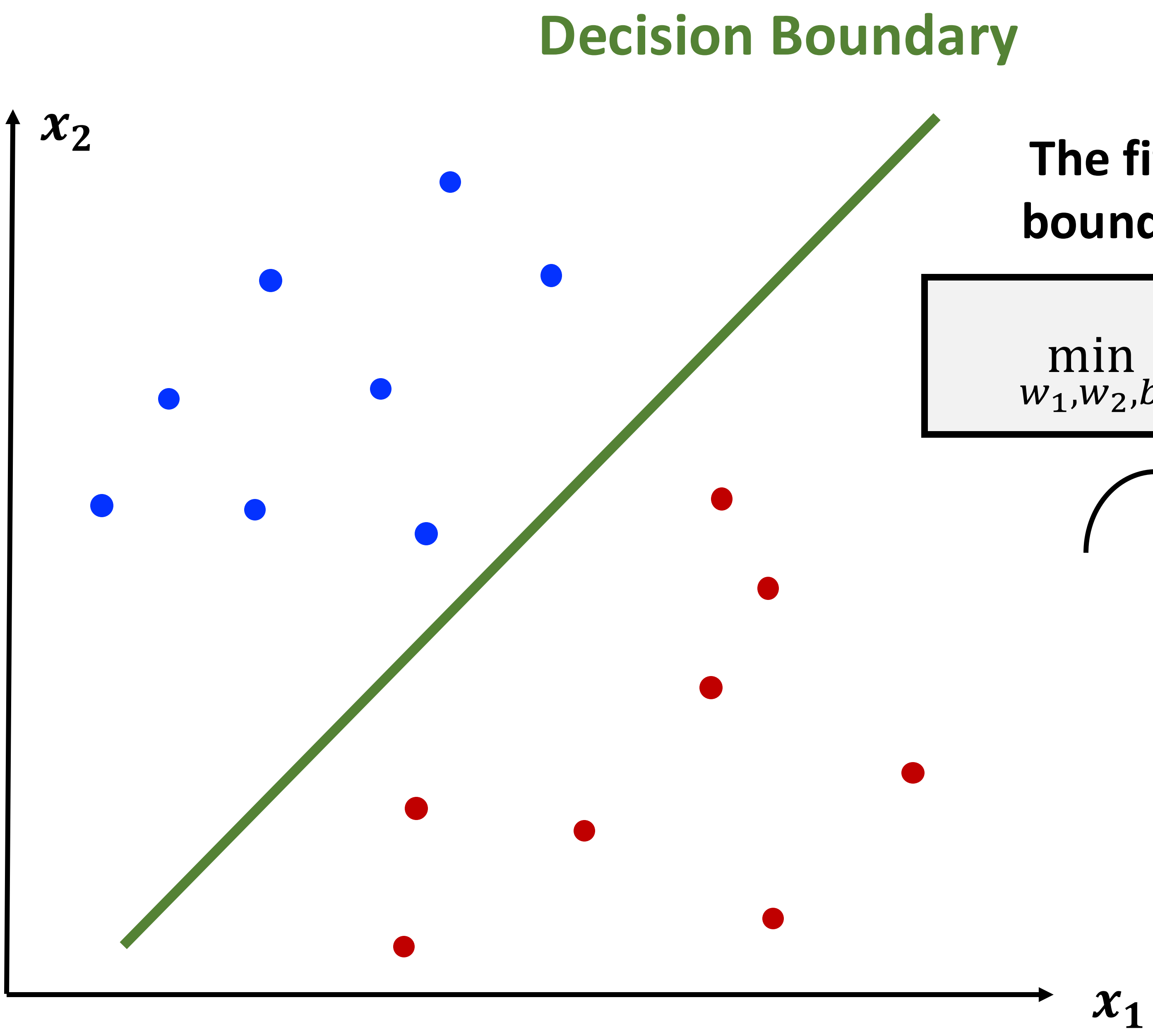


The fitting process finds the best decision boundary that minimizes a "loss function"

$$\boxed{\min_{w_1, w_2, b} \underbrace{\boldsymbol{CrossEntropy}} + \boldsymbol{Regularization}}$$

Measures how good the predicted probability matches the actual label in the training data set

$$\sum_{sample} [-y \log prob - (1-y) \log(1 - prob)]$$



The fitting process finds the best decision boundary that minimizes a “loss function”

$$\min_{w_1,w_2,b} \boldsymbol{CrossEntropy} + \underbrace{\boldsymbol{Regularization}}_{\substack{\text{Helps reduce overfitting} \\ regParam \,*\, \|(w_1, w_2, b)\|^2}}$$

How does fitting really work?

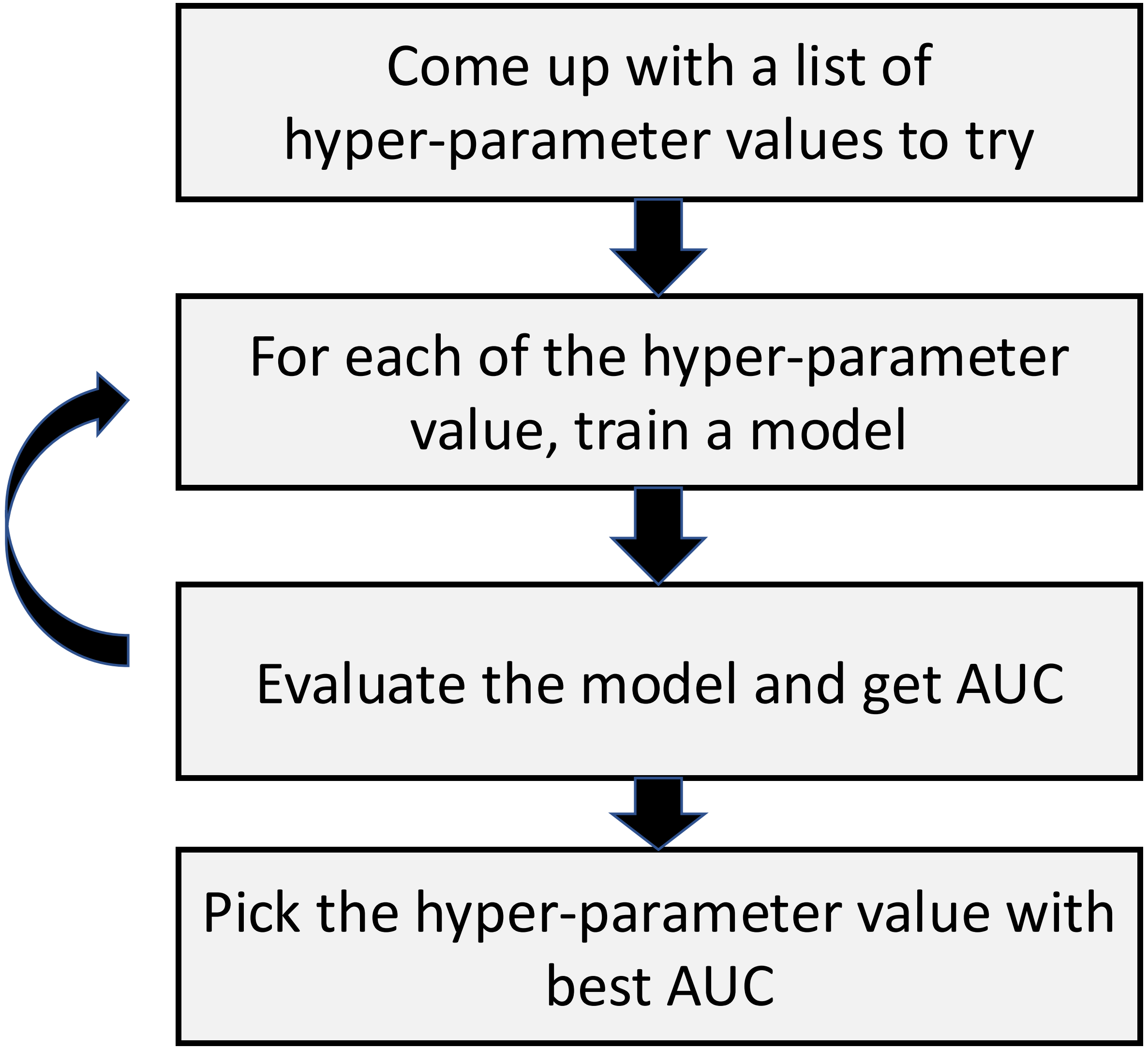
$$\min_{w_1, w_2, b} \boldsymbol{CrossEntropy} + \boldsymbol{Regularization}$$

The fitting process is essentially calling an iterative solver to solve the minimization problem

- **maxIter**: decides how many iterations we run the optimization solver
  - The larger the value, the higher precision we solve the minimization problem
  - If too large, does not improve the precision by much but can slow down fitting process
- **regParam**: controls the size of regularization
  - A small value might help with overfitting
  - If too large, then hurts the accuracy of our model

Some values of the two will lead to a better fitting process, with potentially better AUC.
How to find the right values?

How to find the best hyper-parameter?



How to find the best hyper-parameter?

Come up with a list of hyper-parameter values to try

For each of the hyper-parameter value, train a model

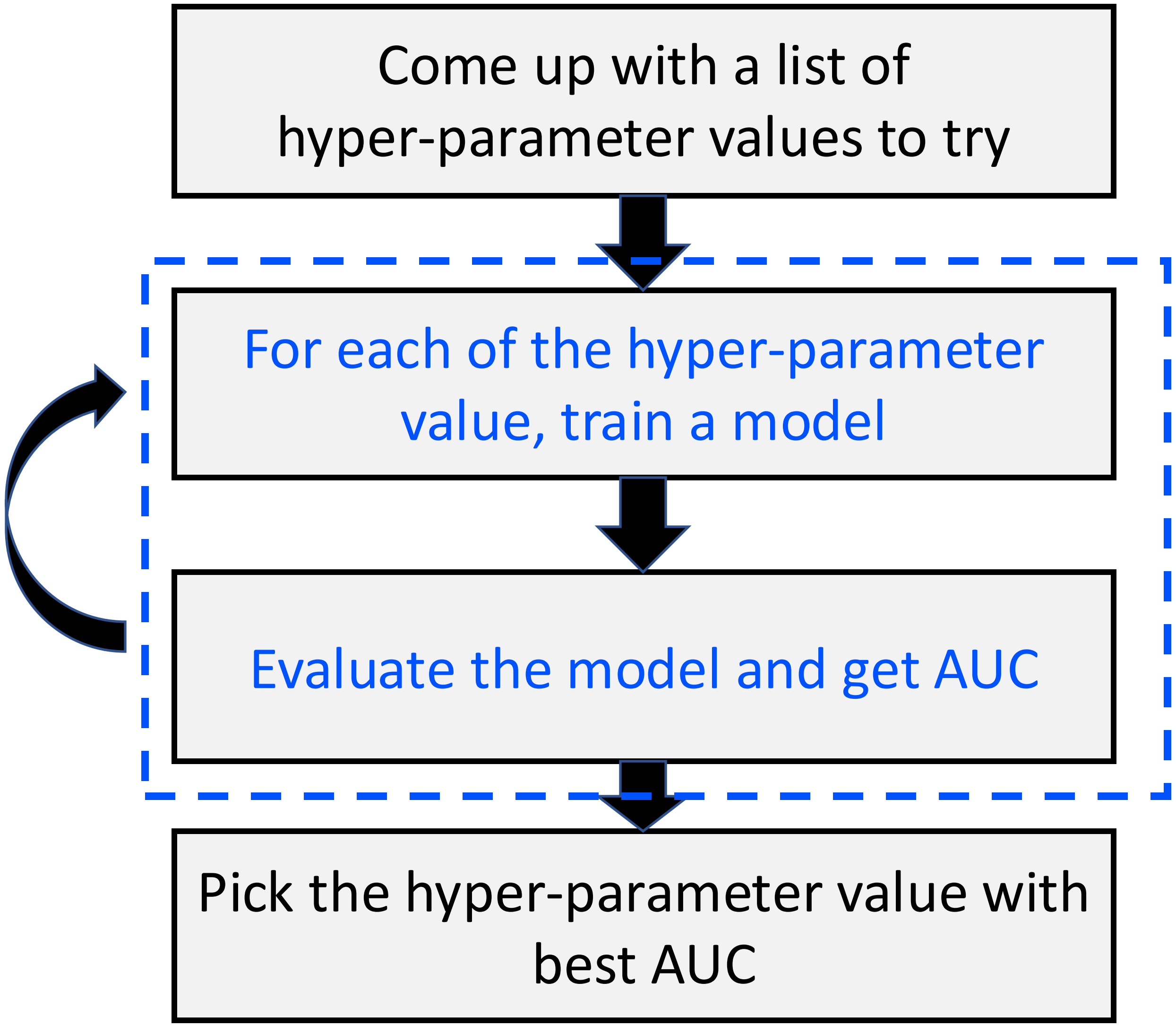
Evaluate the model and get AUC

Pick the hyper-parameter value with best AUC

Typically, for each hyper-parameter, pick a few values, then build parameter grid

		maxIter		
		10	**50**	**100**
regParam	**0.01**	(0.01,10)	(0.01,50)	(0.01,100)
	0.5	(0.5,10)	(0.5,50)	(0.5,100)
	2.0	(2.0,10)	(2.0,50)	(2.0,100)

How to find the best hyper-parameter?



Direct Approach:

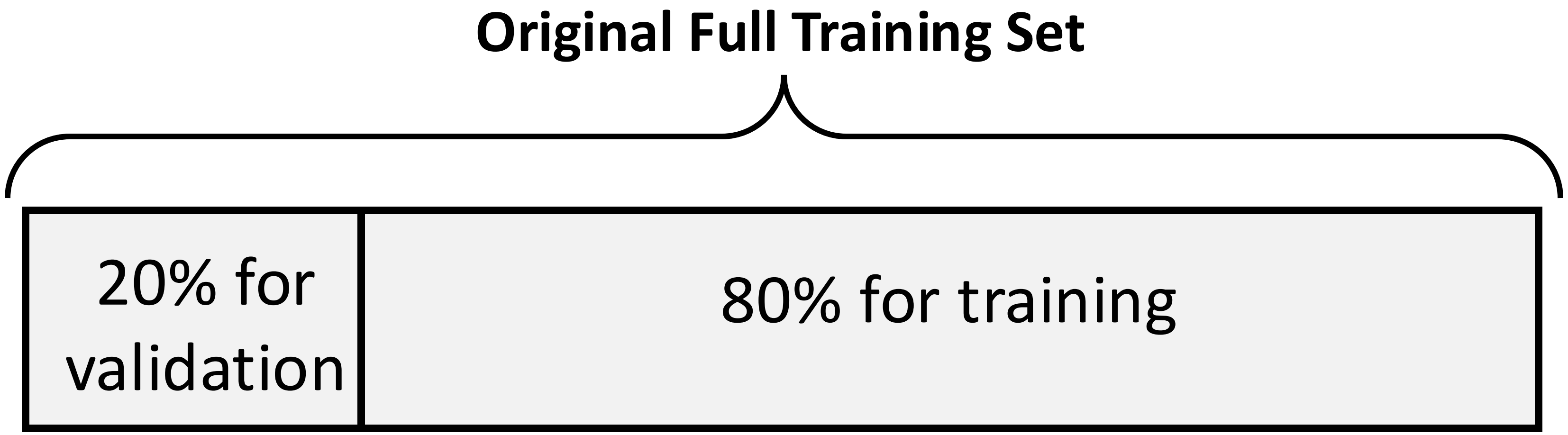
How about train on the training set, and evaluate AUC on the test set?

Not a good idea!
When evaluating the performance of final model, the test set would no longer be reliable as the tuning process has seen the test set before!

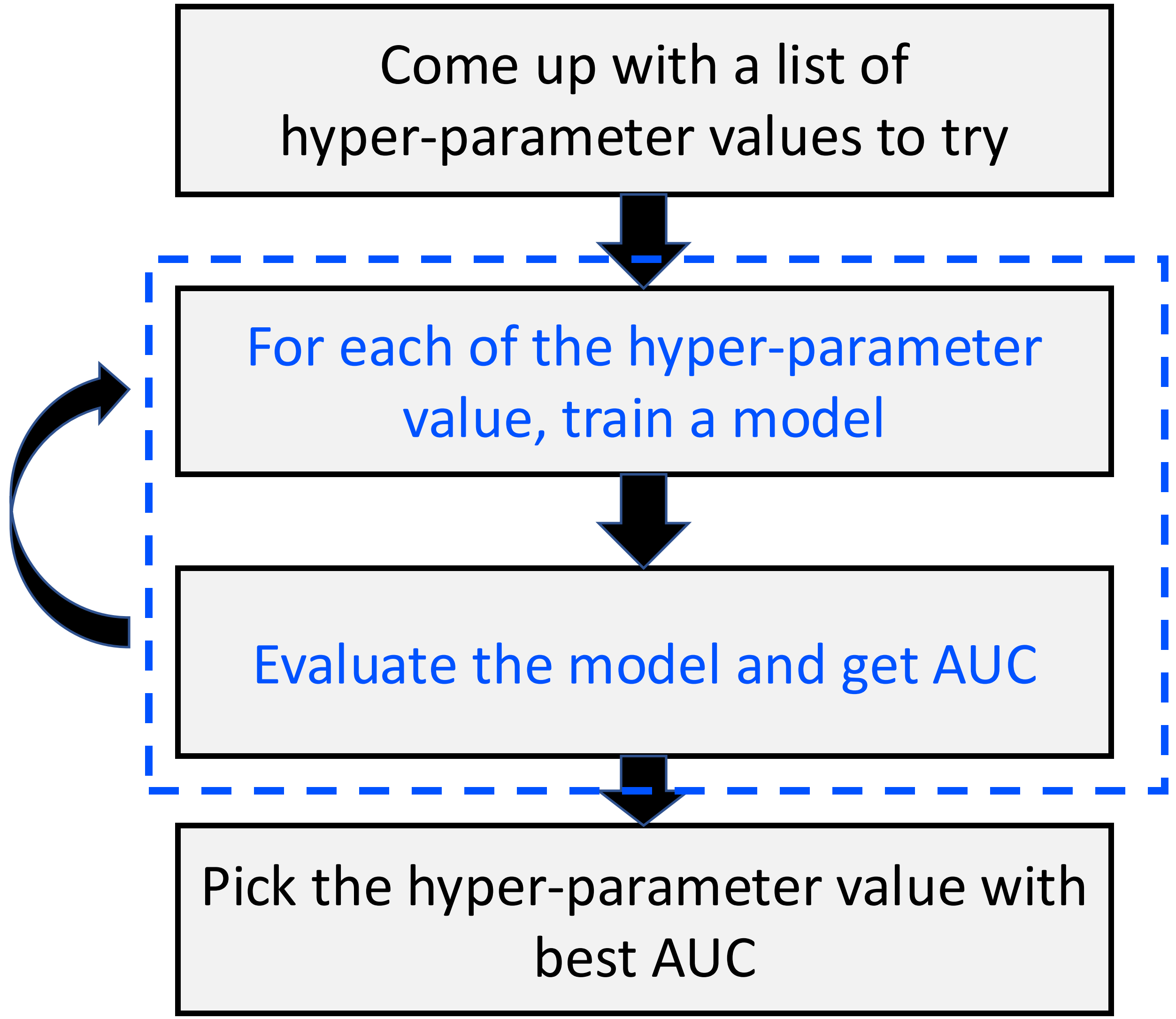
How do we evaluate the trained model without using test data set?

Validation

- We save a portion of training set for evaluation purpose. This is called validation.



How to find the best hyper-parameter?



Train on the 80% subset of training data set.

Evaluate (or validate) on the the remaining 20% of the training data set and get AUC.

An even better idea: cross validation!

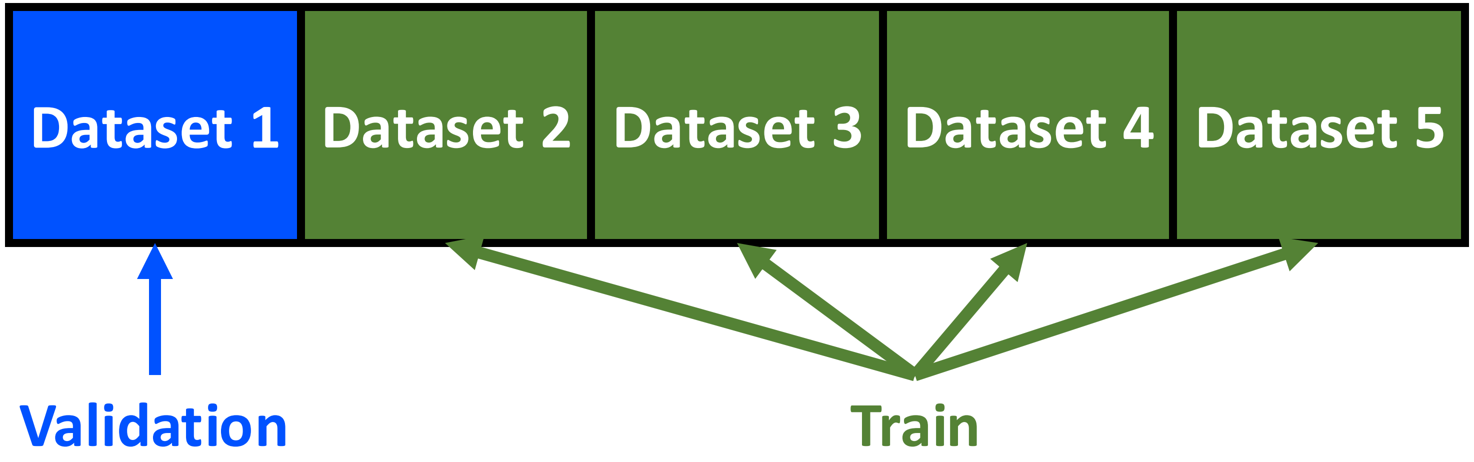
Cross-Validation

Randomly divide full training dataset into 5 pieces

Dataset 1	**Dataset 2**	**Dataset 3**	**Dataset 4**	**Dataset 5**

Cross-Validation

Randomly divide full training dataset into 5 pieces



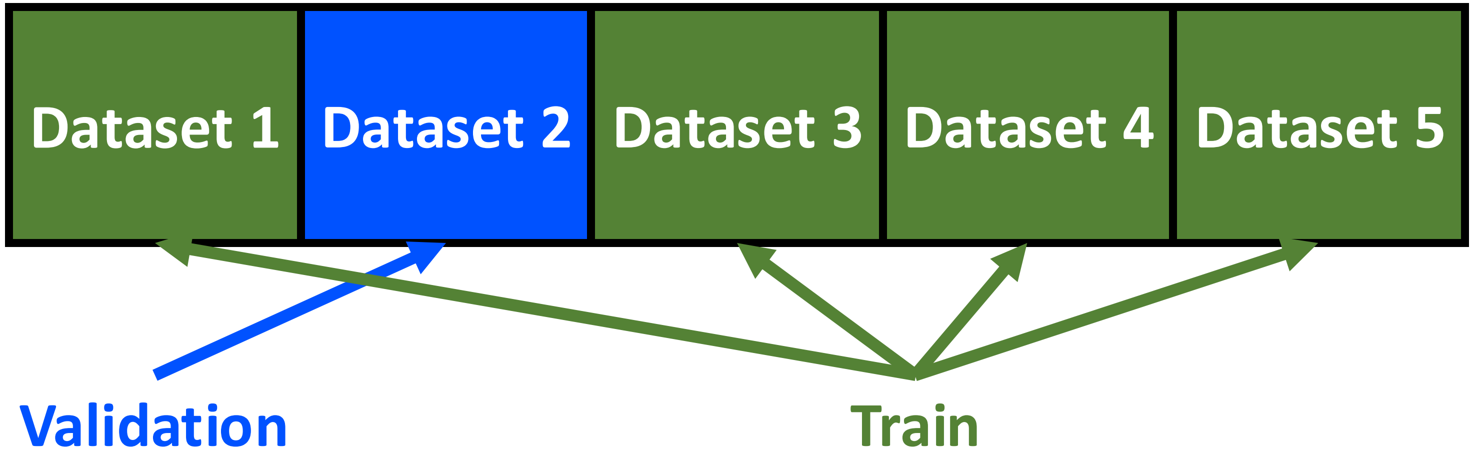
Train model on dataset 2,3,4,5

Evaluate AUC of trained model on dataset 1

$\boldsymbol{AUC_1}$

Cross-Validation

Randomly divide full training dataset into 5 pieces



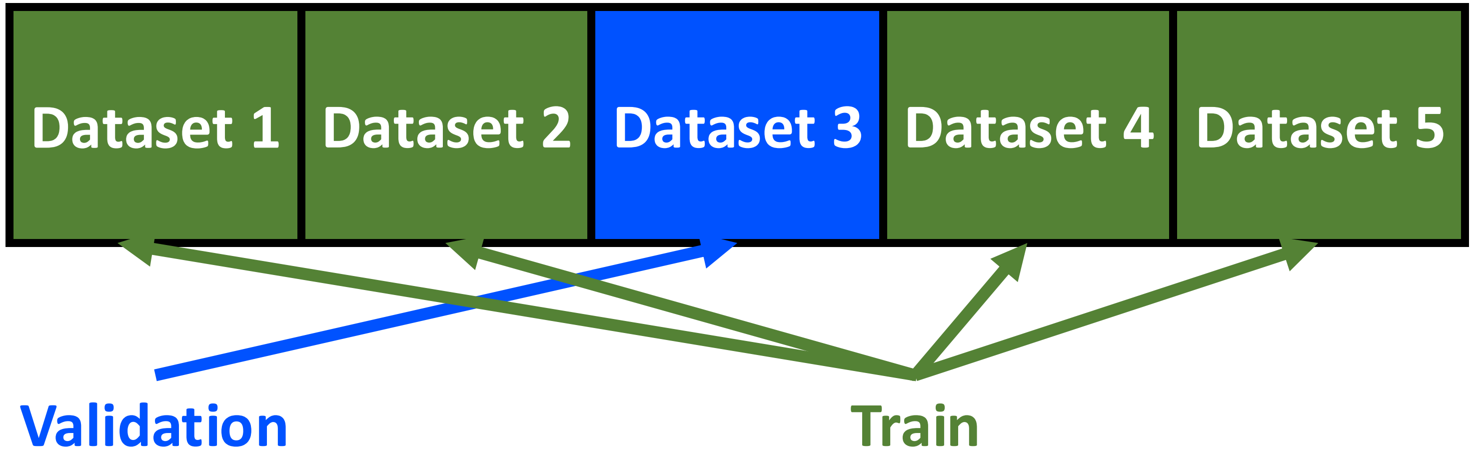
Train model on dataset 1,3,4,5

Evaluate AUC of trained model on dataset 2

AUC_2

Cross-Validation

Randomly divide full training dataset into 5 pieces



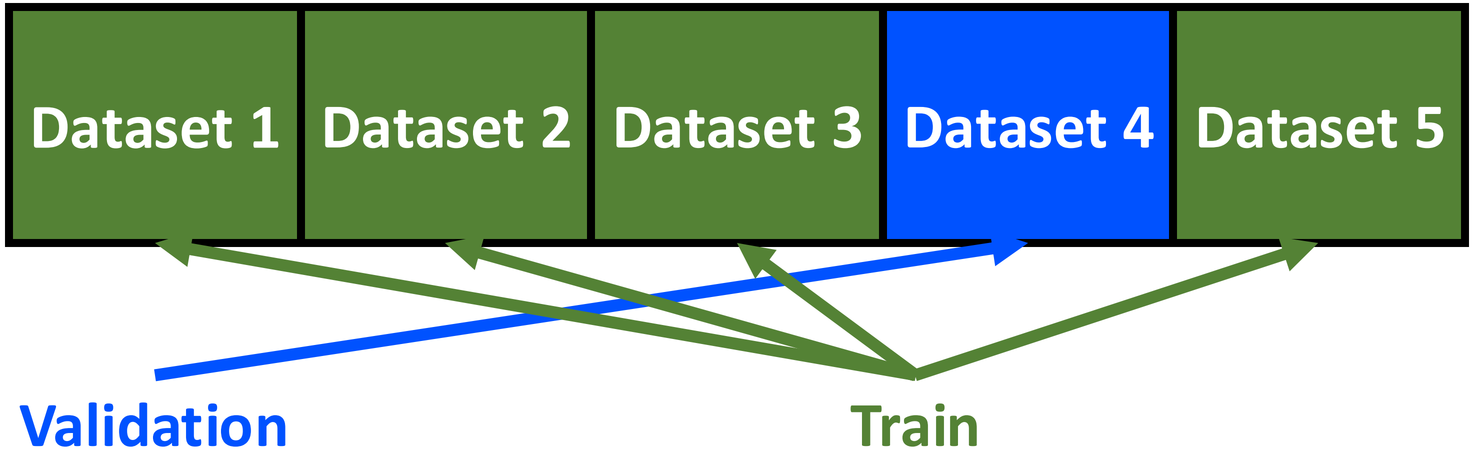
Train model on dataset 1,2,4,5

Evaluate AUC of trained model on dataset 3

$\rightarrow$ AUC_3

Cross-Validation

Randomly divide full training dataset into 5 pieces



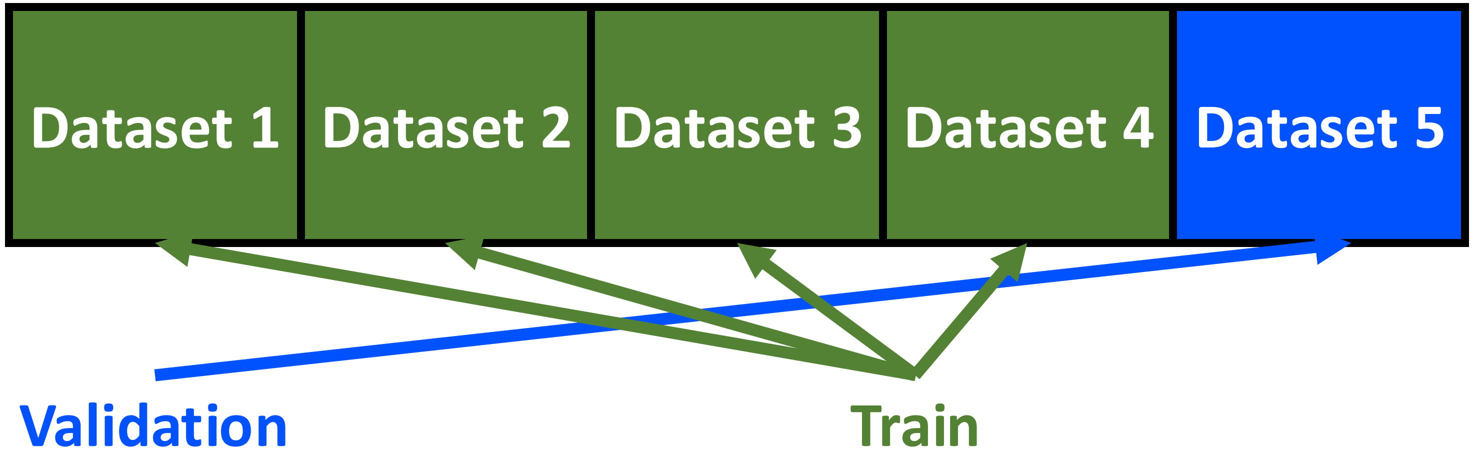
Train model on dataset 1,2,3,5

Evaluate AUC of trained model on dataset 4

$$\boldsymbol{AUC_4}$$

Cross-Validation

Randomly divide full training dataset into 5 pieces

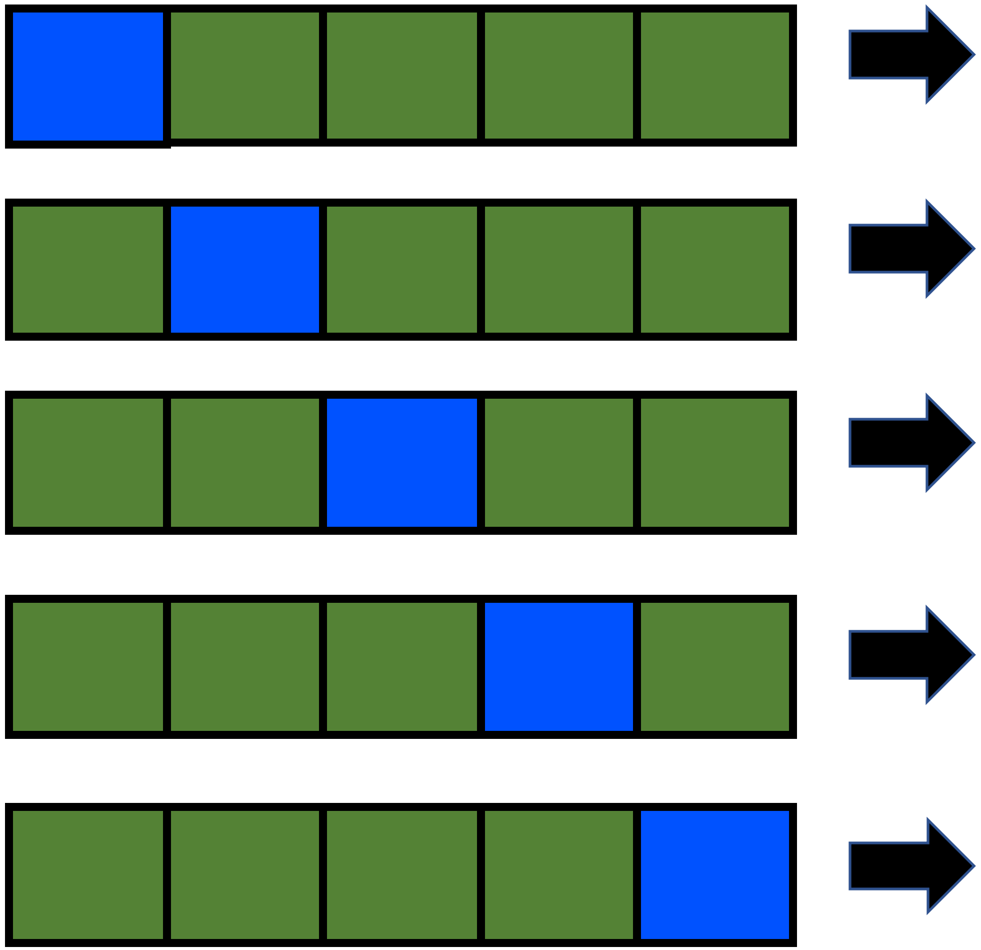


Train model on dataset 1,2,3,4

Evaluate AUC of trained model on dataset 5

AUC_5

Cross-Validation



AUC_1

AUC_2

AUC_3

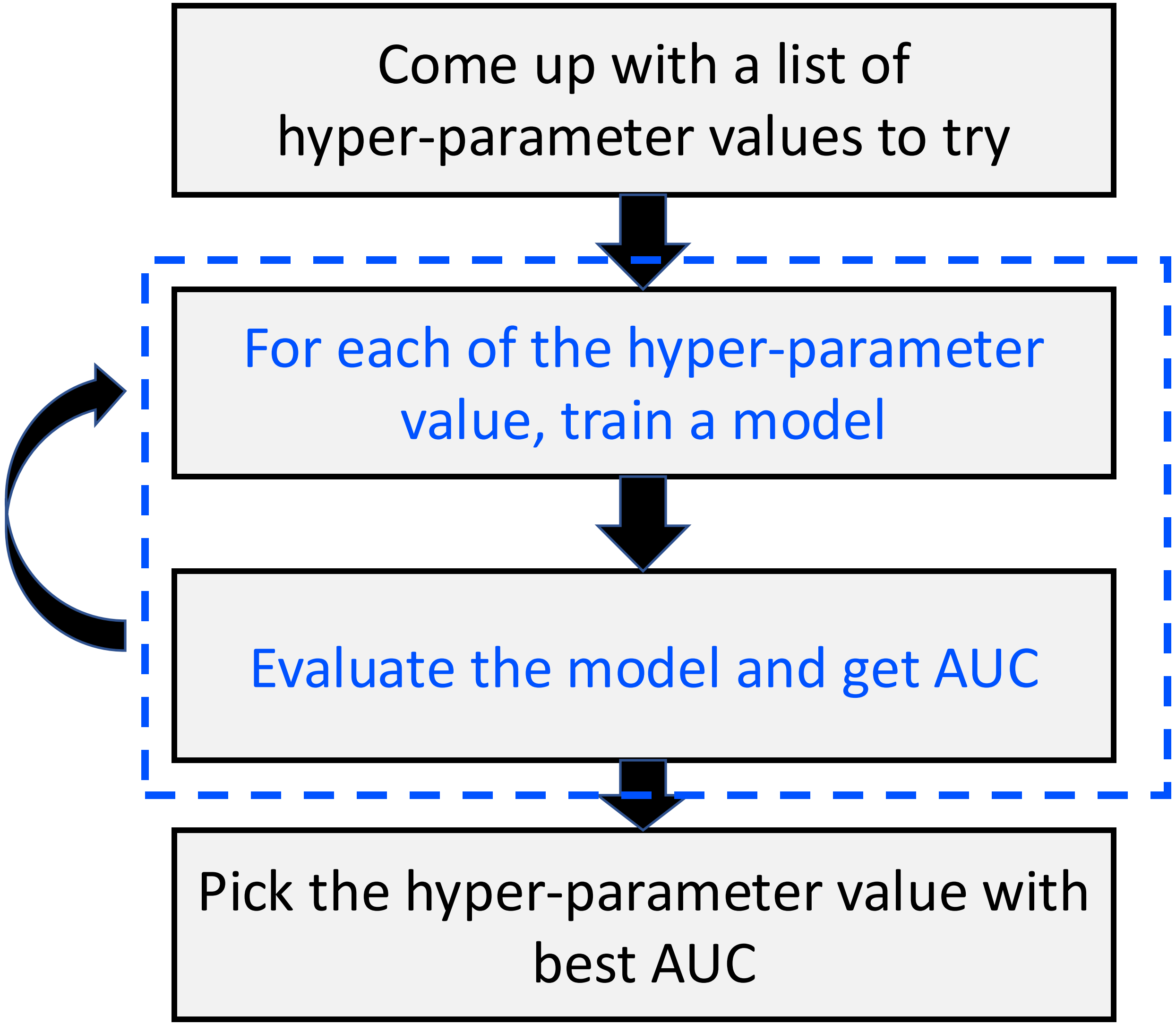
AUC_4

AUC_5

Final AUC for this hyper parameter

$$AUC = \frac{1}{5}(AUC_1 + AUC_2 + AUC_3 + AUC_4 + AUC_5)$$

How to find the best hyper-parameter?



Randomly divide training data set into k equally sized pieces

For $i = 1,2,\ldots,k$

Train on the data set consisting of **all the pieces but $\boldsymbol{i}$**

Evaluate AUC on data set i, get AUC_i

AUC of this hyper-parameter

$$= \frac{\mathbf{1}}{\boldsymbol{k}}(\boldsymbol{AUC_1} + \cdots + \boldsymbol{AUC_k})$$

How do we code cross-validation in SparkML?

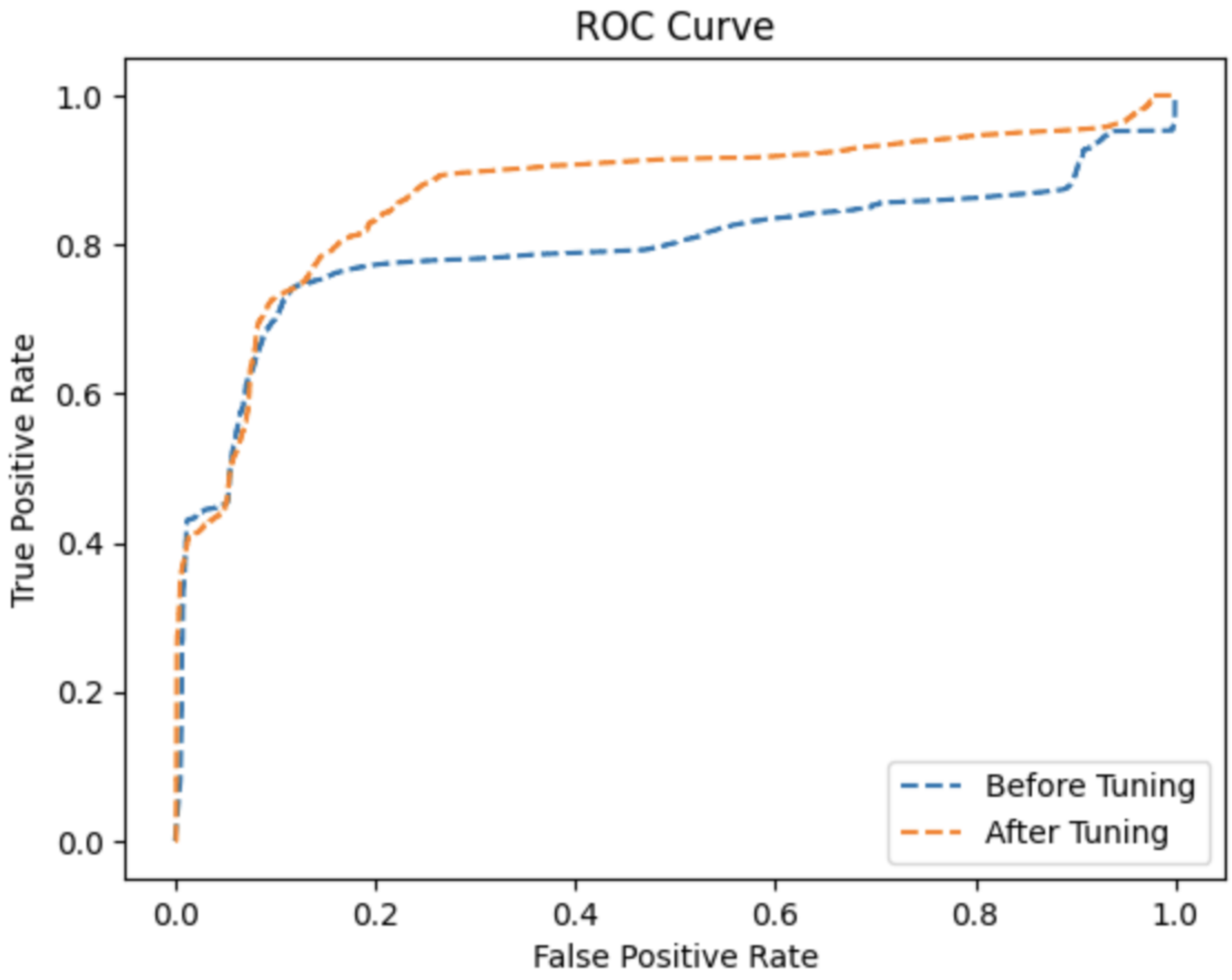
```python
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

lr = LogisticRegression(featuresCol = 'features', labelCol = 'outcome')

# Create ParamGrid for Cross Validation
lr_paramGrid = (ParamGridBuilder()
                .addGrid(lr.regParam, [0.01, 0.5, 2.0])
                .addGrid(lr.maxIter, [1, 5, 10])
                .build())

evaluator = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction',
    labelCol='outcome', metricName='areaUnderROC')

lr_cv = CrossValidator(estimator=lr, estimatorParamMaps=lr_paramGrid,
                    evaluator=evaluator, numFolds=5)
```

How do we code cross-validation in SparkML?

```
lr_cv_model = lr_cv.fit(nslkdd_df)
```

```
lr_cv_prediction_test = lr_cv_model.transform(nslkdd_df_test)
print('Test Area Under ROC (AUC) after Cross-Validation:', evaluator.evaluate(lr_cv_prediction_test))
print('Test Area Under ROC (AUC) before Cross-Validation:', evaluator.evaluate(lr_predictions))
```

```
Test Area Under ROC (AUC) after Cross-Validation: 0.8674445547867702
Test Area Under ROC (AUC) before Cross-Validation: 0.7938144833277767
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Before Tuning
After Tuning

Summary

- ROC Curve and AUC, trading off between true positive and false positive.
- Hyper-parameters of an ML model are the parameters that affect the training/fitting process and the model complexity
- We used cross validation to tune the hyper-parameter to achieve the best AUC