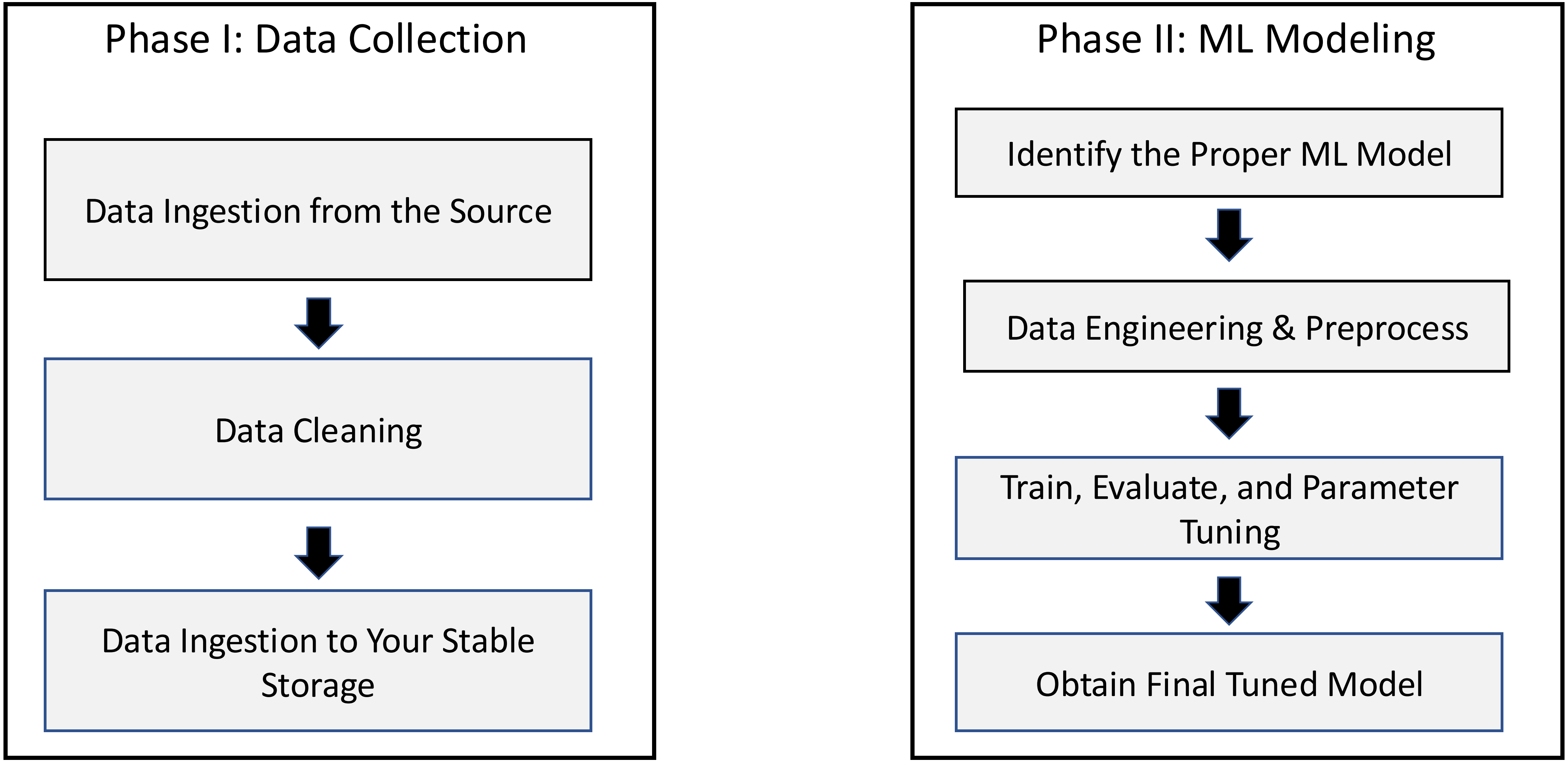
Machine Learning in Spark:
ML Training & Evaluation

Lecture 8 for 14-763/18-763

Guannan Qu

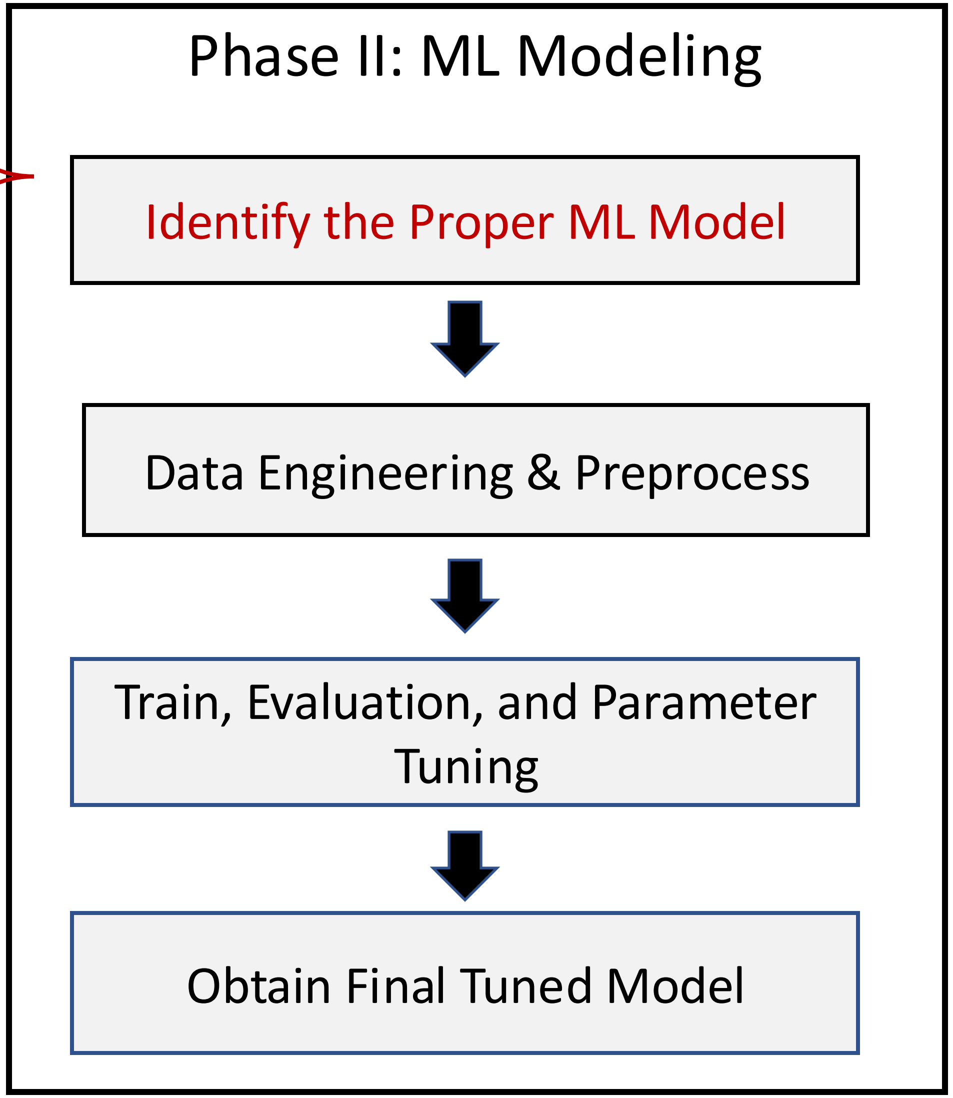
Sept 25, 2024

The Overall ML Process

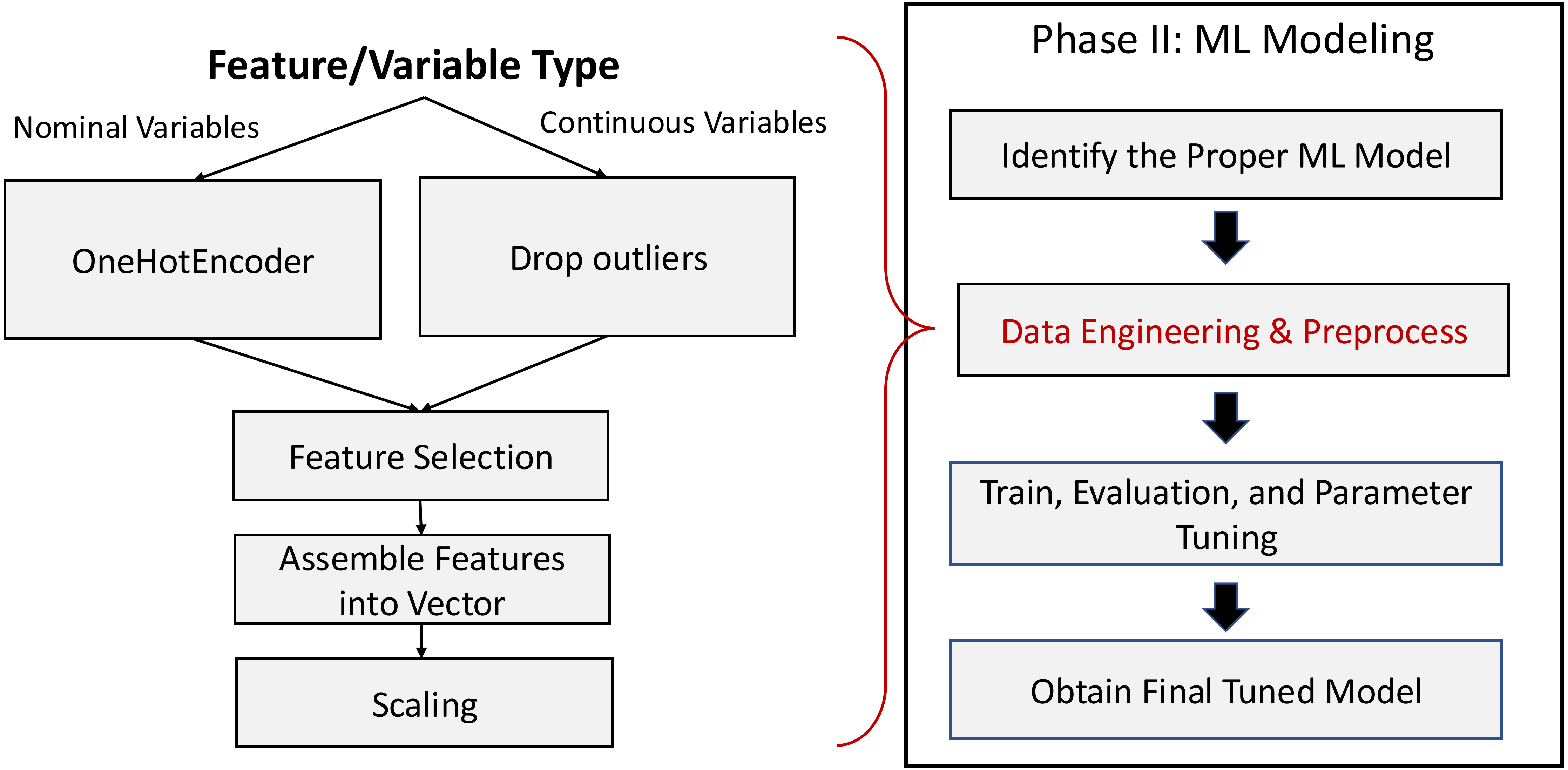


The Overall ML Process

- In the NSL-KDD Example, what is the input and output for ML?
- Classification or regression?
- What are the pros and cons of various ML models?
  - Logistic regression
  - SVM
  - Decision tree
  - Naïve bayes
  - Others…

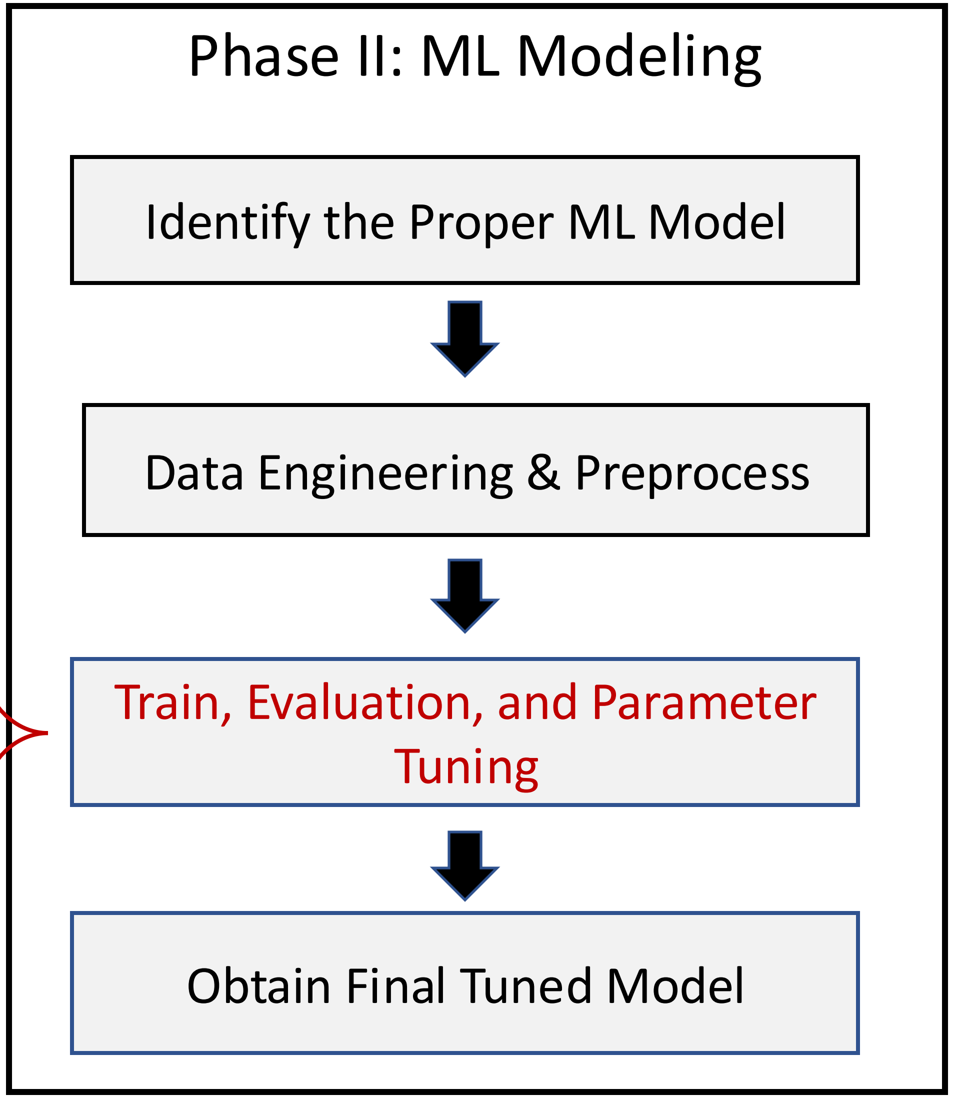


The Overall ML Process



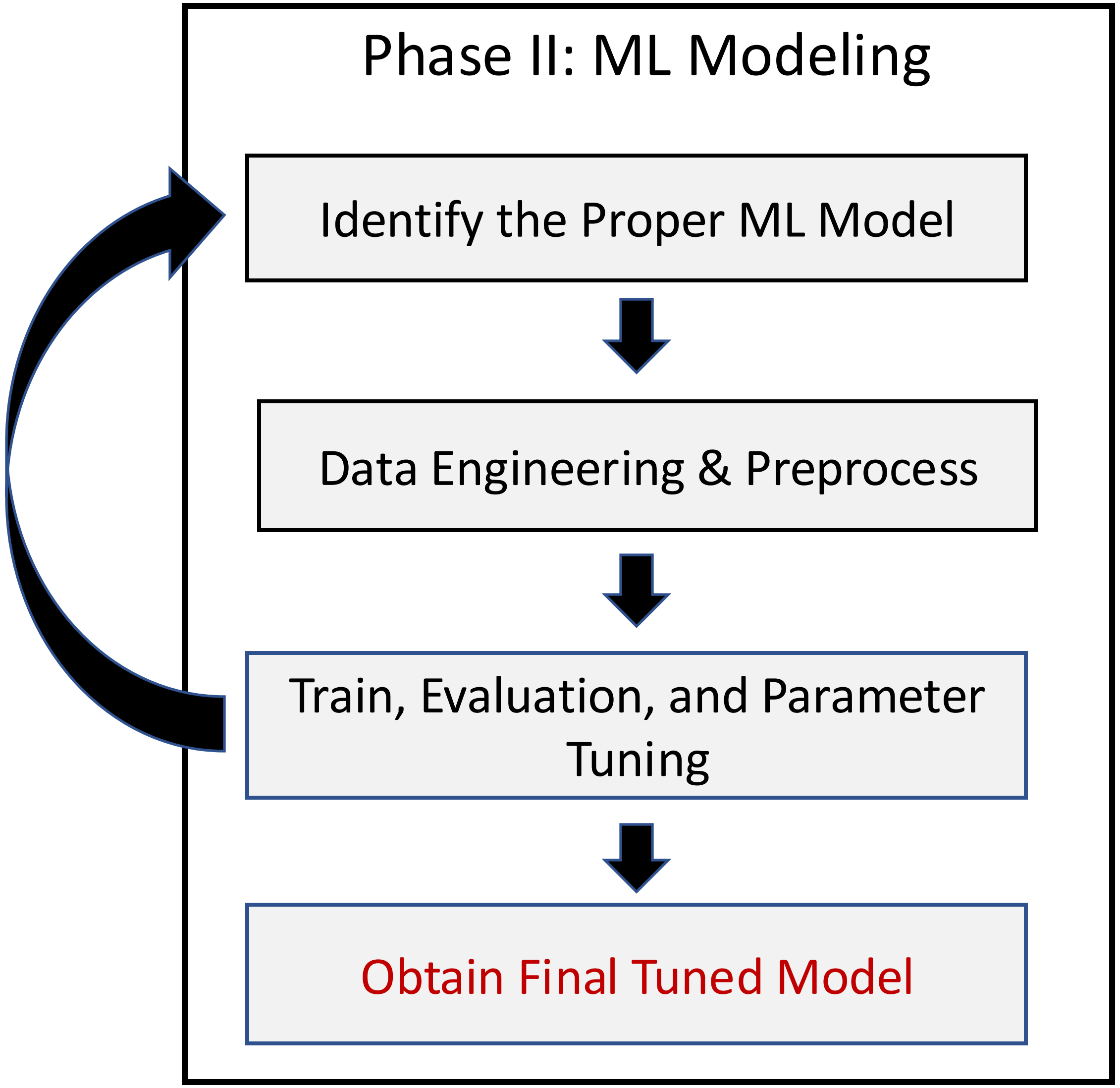
The Overall ML Process

- How to train the model?
- How do we know our model is good?
  - Accuracy
  - ROC/AUC
  - Mean Square Error
  - …
- Train vs Validate vs Test
- Parameter Tuning via Cross-validation



The Overall ML Process

May need to try a few different models



Today's Agenda

- Review fundamental concepts of machine learning
  - Logistic regression – the first ML model we will try for the NSL-KDD dataset

- ML Process with SparkML using Logistic Regression

Introduction of ML

How do we use **info about each connection** to predict

whether there is attack?

the type of attack? (homework)

1	Duration	Continuous	Integers	0 - 54451
2	Protocol Type	Categorical	Strings	
3	Service	Categorical	Strings	
…	…	…	…	…
40	Dst Host Rerror Rate	Discrete	Floats (hundredths of a decimal)	0 - 1
41	Dst Host Srv Rerror Rate	Discrete	Floats (hundredths of a decimal)	0 - 1

Introduction of ML

How do we use **info about each connection** to predict

Duration, Protocol Type, … (41 features)

whether there is attack?

the type of attack? (homework)

Input/Feature → ML Model → Output/Target/Label

Regression vs Classification

Classification: the output/target can only take a finite set of values

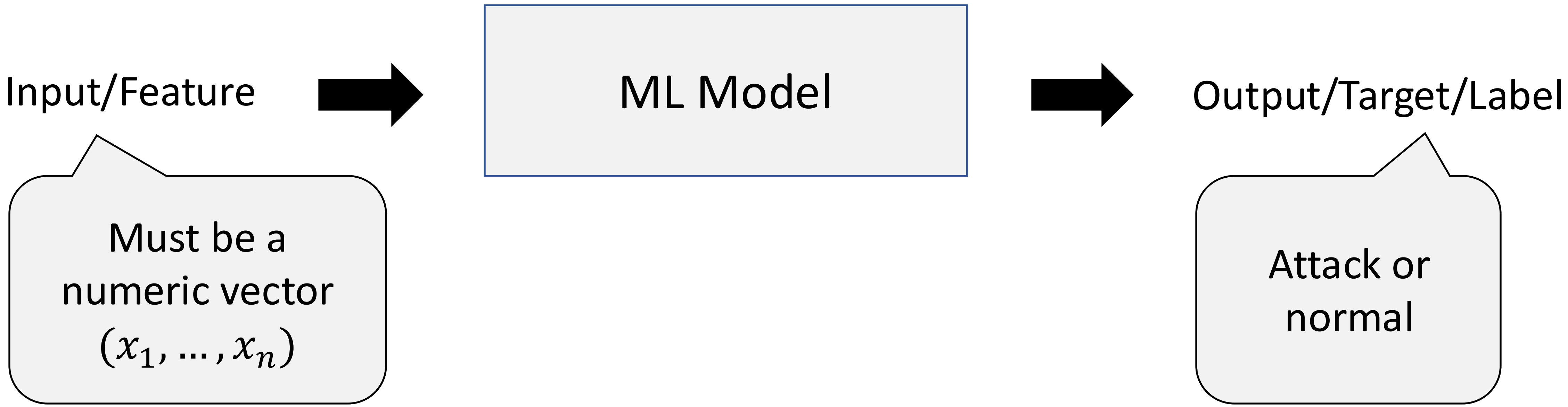
- Normal/Attack – binary classification
- Normal/DoS/Probe/U2R/R2L – multiclass classification

Regression: the output/target is a numeric and continuous variable

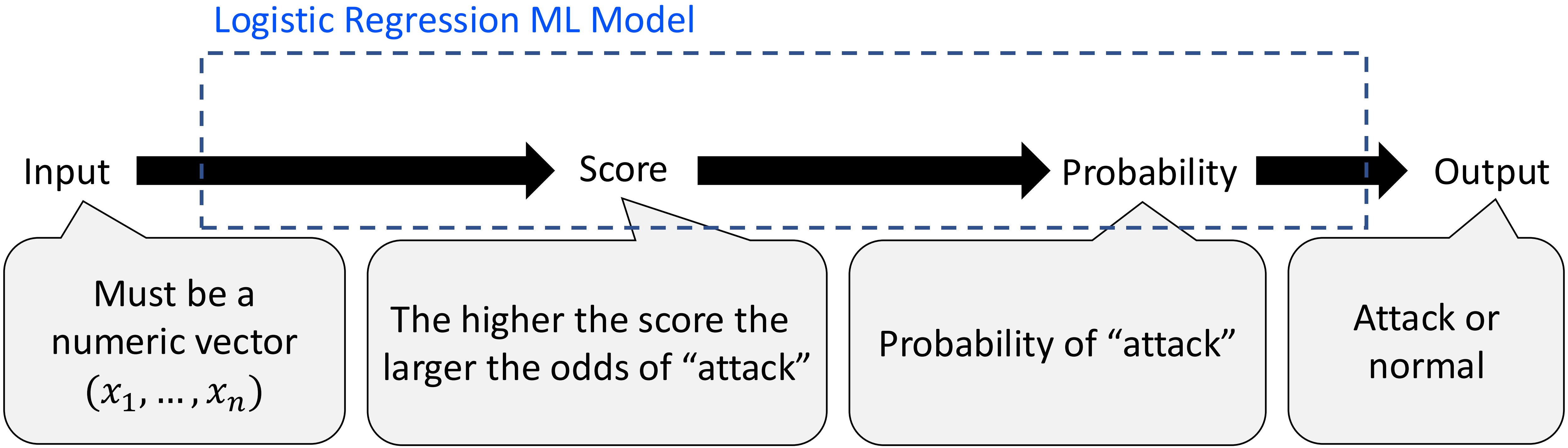
- Predict a housing price

Logistic Regression

- Despite the name, logistic regression is NOT for regression!
- It is one of the simplest ML model for binary classification…
- …and also one of the most frequently asked ML models during job interviews



Logistic Regression



Logistic Regression

Linear function

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

Input → Score → Probability → Output

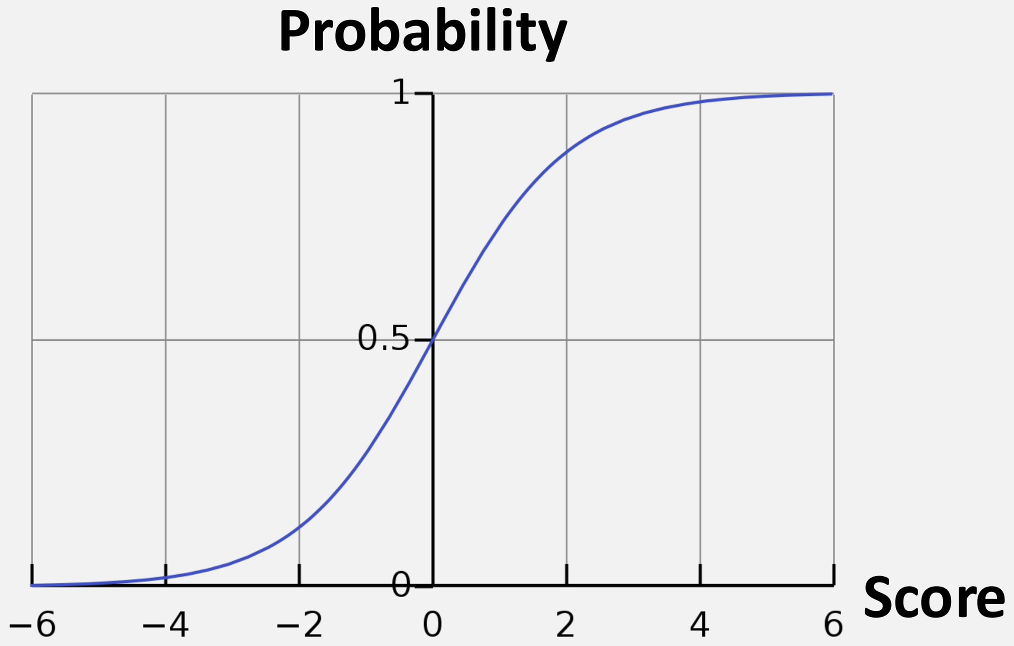
Must be a numeric vector $(x_1, \ldots, x_n)$

The higher the score the larger the odds of “attack”

Logistic Re

Logistic Function

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



Input → Score → Probability → Output

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

Probability of "attack"

Logistic Regression

Thresholding

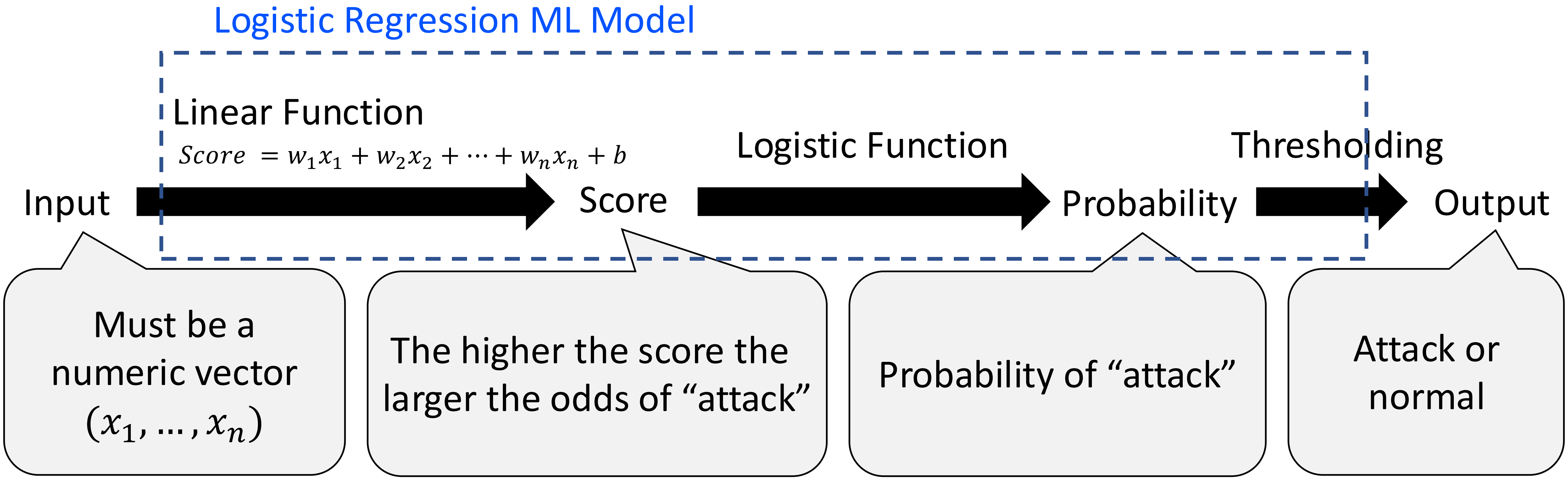
If prob. > threshold, then predict ”attack”.
Otherwise predict “normal”.
Default threshold is 0.5

Input → Score → Probability → Output

Probability of “attack”

Attack or normal

Logistic Regression



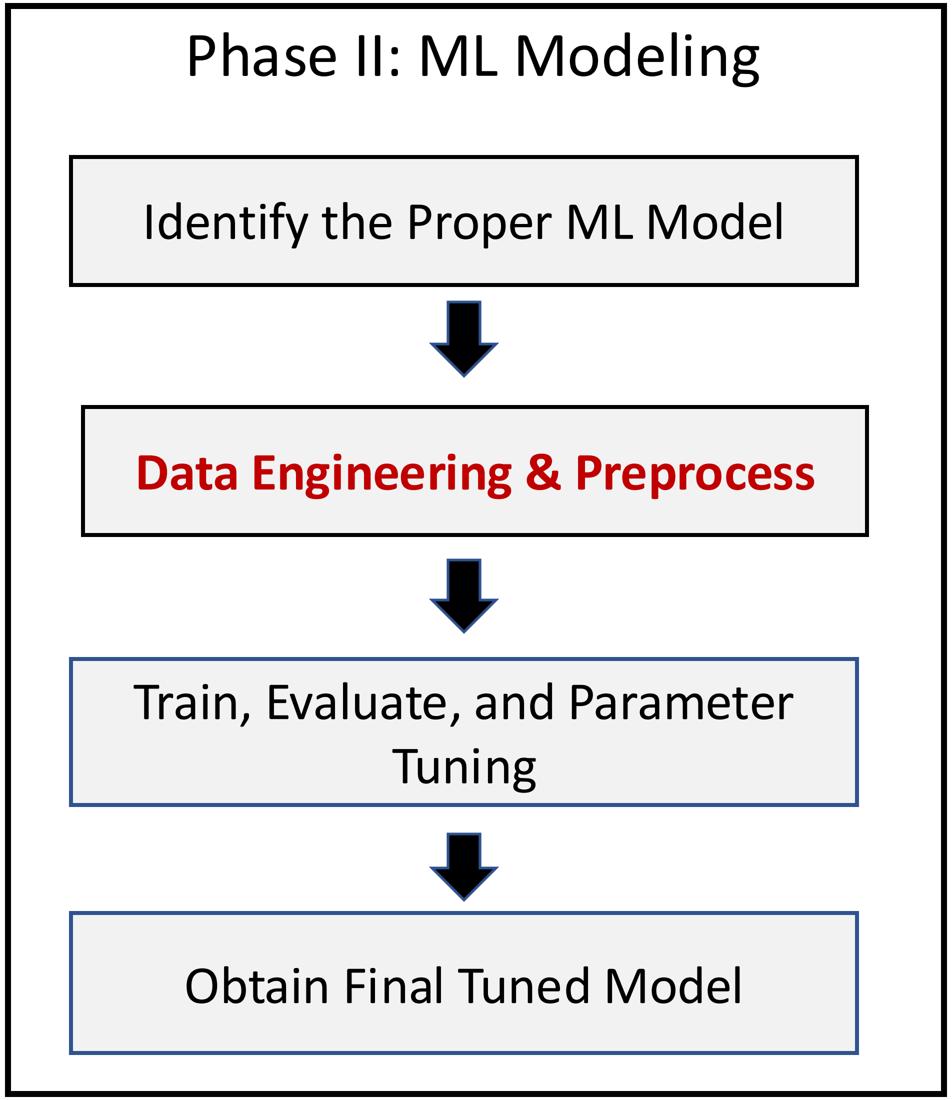
Train vs Test

- Training/fitting: given a training dataset of input-output pairs, find the best logistic regression weights for this prediction task.
- Test: use the trained model to make prediction on unseen data

Today’s Agenda

- Review fundamental concepts of machine learning
  - What is input & output of ML model
  - Classification vs Regression
  - Logistic regression – the first ML model we will try for the NSL-KDD dataset

- ML Process with SparkML using Logistic Regression


Phase II: ML Modeling
Identify the Proper ML Model
Data Engineering & Preprocess
Train, Evaluate, and Parameter Tuning
Obtain Final Tuned Model

Recall what we did

We decided to conduct the following steps on our dataset

- Cast columns as appropriate types (particularly numerical columns)
- StringIndexer and OneHotEncoder for nominal columns
- Throwing away 6 columns with high correlation coefficients
- Assemble features into vector and scaling

Today, we are going to use both KDDTrain+.txt and KDDTest+.txt
Do we need to repeat the above procedure for KDDTest+.txt?

PySpark provides the concept of “Transformer” and ”Pipeline” that standardizes dataframe processing and can be reused!

Pipeline

- Transformer: takes in a dataframe and produce a dataframe
  - Key method: transform()
  - Example: VectorAssembler

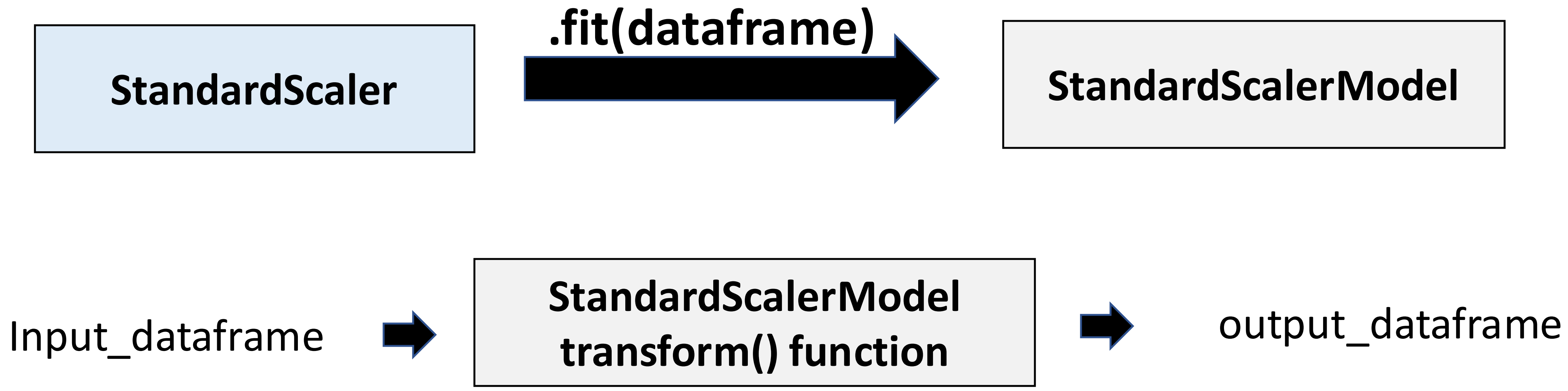
Input_dataframe

VectorAssembler
transform() function

output_dataframe

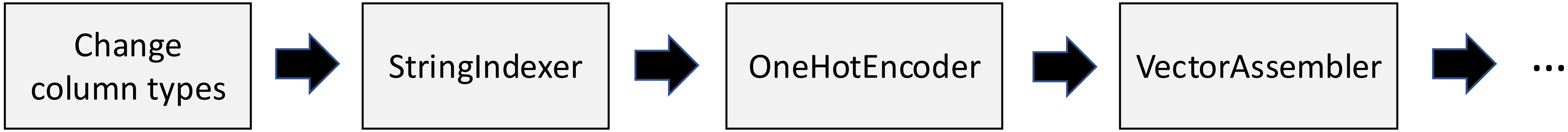
Pipeline

- Estimator: produces a transformer after fitting
  - Key method: fit()
  - Example: StandardScaler



Pipeline

Pipeline: stages of transformers or estimators connected together



```python
pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_onehot_encoder,
    stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
```

```python
stage_typecaster = FeatureTypeCaster()
```

```python
class FeatureTypeCaster(Transformer): # this transformer will cast the columns as appropriate types
    def __init__(self):
        super().__init__()

    def _transform(self, dataset):
        output_df = dataset
        for col_name in binary_cols + continuous_cols:
            output_df = output_df.withColumn(col_name,col(col_name).cast(DoubleType()))

        return output_df
```

```
pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_onehot_encoder,
    stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
```

```
# Stage where nominal columns are transformed to index columns using StringIndexer
nominal_id_cols = [x+"_index" for x in nominal_cols]
nominal_onehot_cols = [x+"_encoded" for x in nominal_cols]
stage_nominal_indexer = StringIndexer(inputCols = nominal_cols, outputCols = nominal_id_cols )

# Stage where the index columns are further transformed using OneHotEncoder
stage_nominal_onehot_encoder = OneHotEncoder(inputCols=nominal_id_cols, outputCols=nominal_onehot_cols)
```

```
pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_onehot_encoder,
    stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
```

```
# Stage where all relevant features are assembled into a vector (and dropping a few)
feature_cols = continuous_cols+binary_cols+nominal_onehot_cols
corelated_cols_to_remove = ["dst_host_serror_rate","srv_serror_rate","dst_host_srv_serror_rate",
                    "srv_rerror_rate","dst_host_rerror_rate","dst_host_srv_rerror_rate"]
for col_name in corelated_cols_to_remove:
    feature_cols.remove(col_name)
stage_vector_assembler = VectorAssembler(inputCols=feature_cols, outputCol="vectorized_features")
```

```
pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_onehot_encoder,
    stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
```

```
# Stage where we scale the columns
stage_scaler = StandardScaler(inputCol= 'vectorized_features', outputCol= 'features')
```

```python
pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_onehot_encoder,
    stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
```

```python
# Stage for creating the outcome column representing whether there is attack
stage_outcome = OutcomeCreater()
```

```python
class OutcomeCreater(Transformer): # this defines a transformer that creates the outcome column

    def __init__(self):
        super().__init__()

    def _transform(self, dataset):
        label_to_binary = udf(lambda name: 0.0 if name == 'normal' else 1.0)
        output_df = dataset.withColumn('outcome', label_to_binary(col('class'))).drop("class")
        output_df = output_df.withColumn('outcome', col('outcome').cast(DoubleType()))
        output_df = output_df.drop('difficulty')
        return output_df
```

```python
pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_onehot_encoder,
    stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
```

```python
# Removing all unnecessary columbs, only keeping the 'features' and 'outcome' columns
stage_column_dropper = ColumnDropper(columns_to_drop = nominal_cols+nominal_id_cols+
    nominal_onehot_cols+ binary_cols + continuous_cols + ['vectorized_features'])
```

```python
class ColumnDropper(Transformer): # this transformer drops unnecessary columns
    def __init__(self, columns_to_drop = None):
        super().__init__()
        self.columns_to_drop=columns_to_drop
    def _transform(self, dataset):
        output_df = dataset
        for col_name in self.columns_to_drop:
            output_df = output_df.drop(col_name)
        return output_df
```

```
spark = SparkSession.builder \
    .master("local[*]") \
    .appName("GenericAppName") \
    .getOrCreate()

nslkdd_raw = spark.read.csv('./NSL-KDD/KDDTrain+.txt',header=False).toDF(*col_names)
nslkdd_test_raw = spark.read.csv('./NSL-KDD/KDDTest+.txt',header=False).toDF(*col_names)

preprocess_pipeline = get_preprocess_pipeline()
preprocess_pipeline_model = preprocess_pipeline.fit(nslkdd_raw)

nslkdd_df = preprocess_pipeline_model.transform(nslkdd_raw)
nslkdd_df_test = preprocess_pipeline_model.transform(nslkdd_test_raw)
```

Read raw dataframe from file

Get the pipeline

Note: need to fit pipeline to training data set nslkdd_raw

Let's check out the transformed dataframes we just obtained!

```
nslkdd_df.printSchema()
nslkdd_df.show(1)
```

✓ 0.9s

```
root
 |-- features: vector (nullable = true)
 |-- outcome: double (nullable = true)

+--------------------+-------+
|            features|outcome|
+--------------------+-------+
|(113,[1,13,14,17,...|    0.0|
+--------------------+-------+
only showing top 1 row
```

```
nslkdd_df_test.printSchema()
nslkdd_df_test.show(1)
```

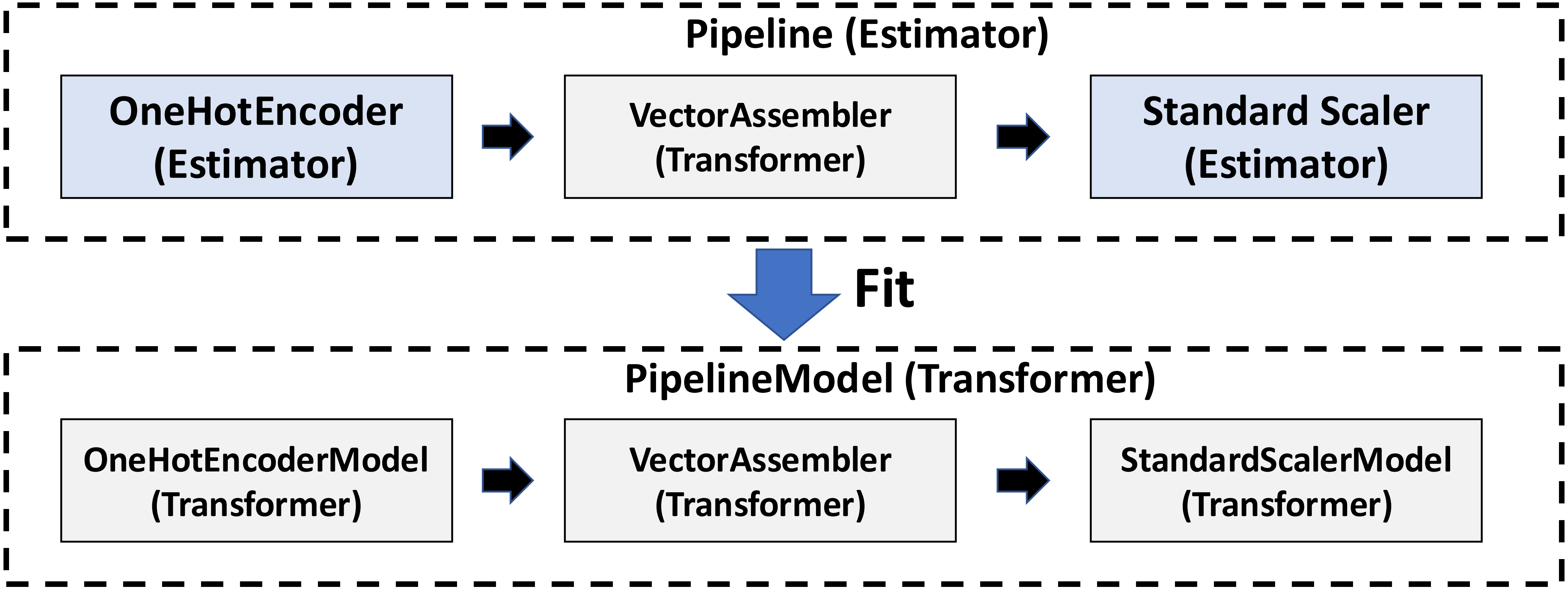
✓ 0.3s

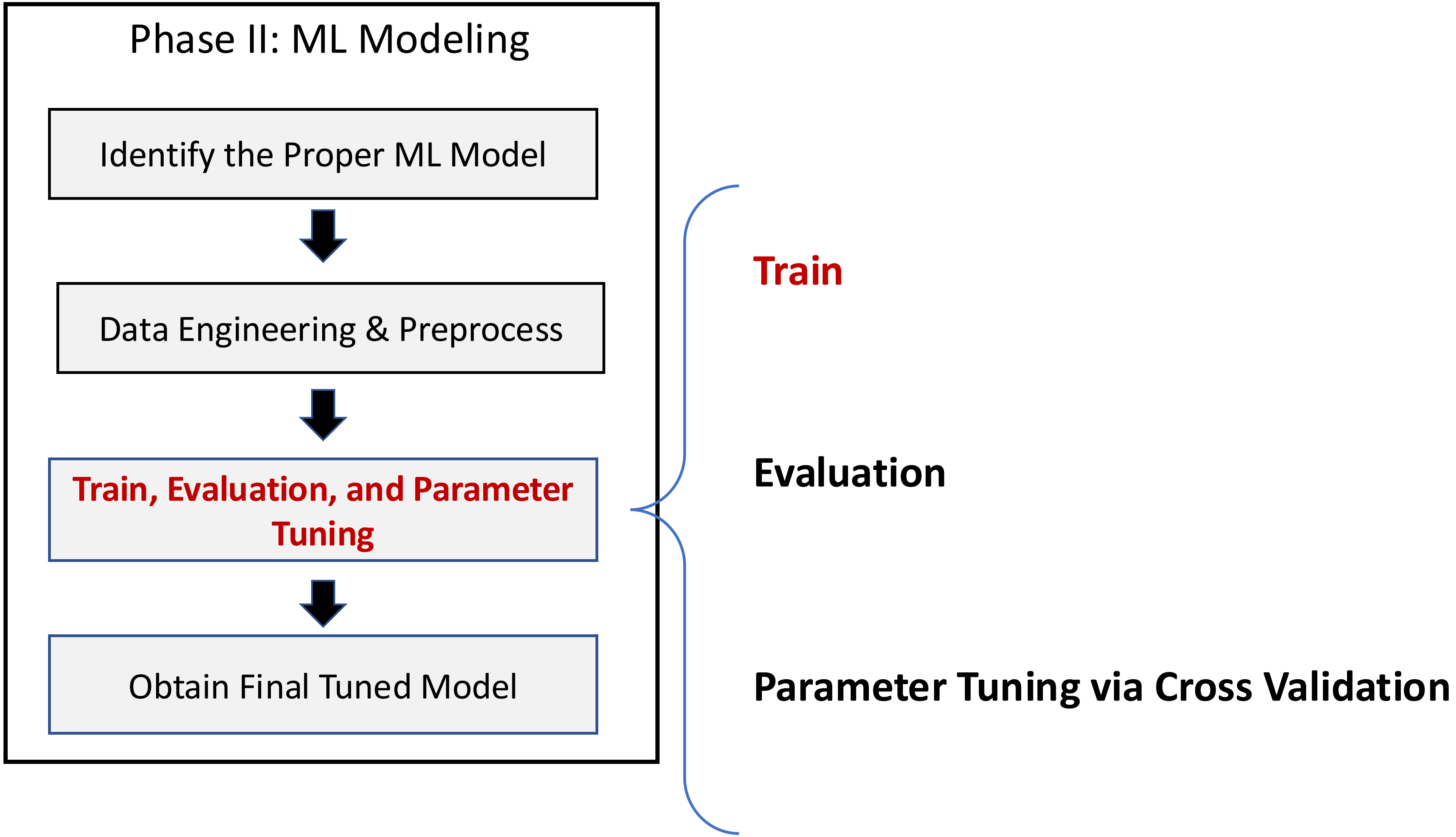
```
root
 |-- features: vector (nullable = true)
 |-- outcome: double (nullable = true)

+--------------------+-------+
|            features|outcome|
+--------------------+-------+
|(113,[13,14,16,17...|    1.0|
+--------------------+-------+
only showing top 1 row
```

```
pipeline = Pipeline(stages=[stage_typecaster,stage_nominal_indexer,stage_nominal_onehot_encoder,
    stage_vector_assembler,stage_scaler,stage_outcome,stage_column_dropper])
```

What is the type of Pipeline?

- The Pipeline itself is an estimator and needs to be fitted.
- When calling pipeline.fit(…), what happens is that the fit method is called for all estimator stages in the pipeline.




Phase II: ML Modeling
Identify the Proper ML Model
Data Engineering & Preprocess
Train, Evaluation, and Parameter Tuning
Obtain Final Tuned Model
Train
Evaluation
Parameter Tuning via Cross Validation

Train

```
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
```

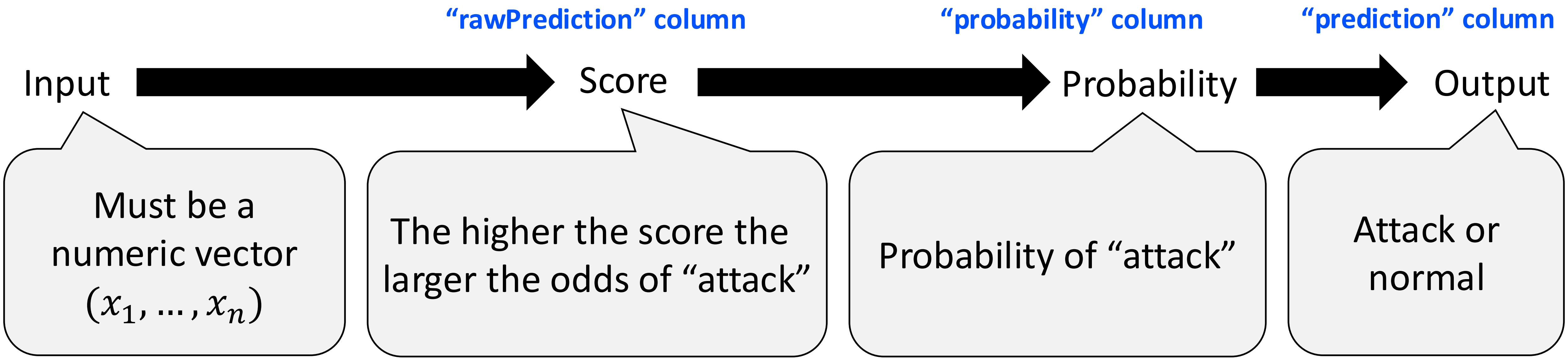
How to use the trained model to make prediction?

```
predictions = lrModel.transform(nslkdd_df_test)
```

```python
predictions.printSchema()
```

✓ 0.6s

```
root
 |-- features: vector (nullable = true)
 |-- outcome: double (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```



```python
predictions.select("rawPrediction","probability","prediction","outcome").toPandas().head()
```

✓ 1.4s

	rawPrediction	probability	prediction	outcome
0	[-8.500778162324389, 8.500778162324389]	[0.00020326877258515 72, 0.9997967312274149]	1.0	1.0
1	[-6.916372280772924, 6.916372280772924]	[0.0009904380780218754, 0.9990095619219781]	1.0	1.0
2	[3.4433803926294946, -3.4433803926294946]	[0.9690331154391638, 0.030966884560836183]	0.0	0.0
3	[-3.416196864675788, 3.416196864675788]	[0.0317930893592043, 0.9682069106407957]	1.0	1.0
4	[3.267438385406158, -3.267438385406158]	[0.9632947055293201, 0.03670529447067994]	0.0	1.0

This is estimator

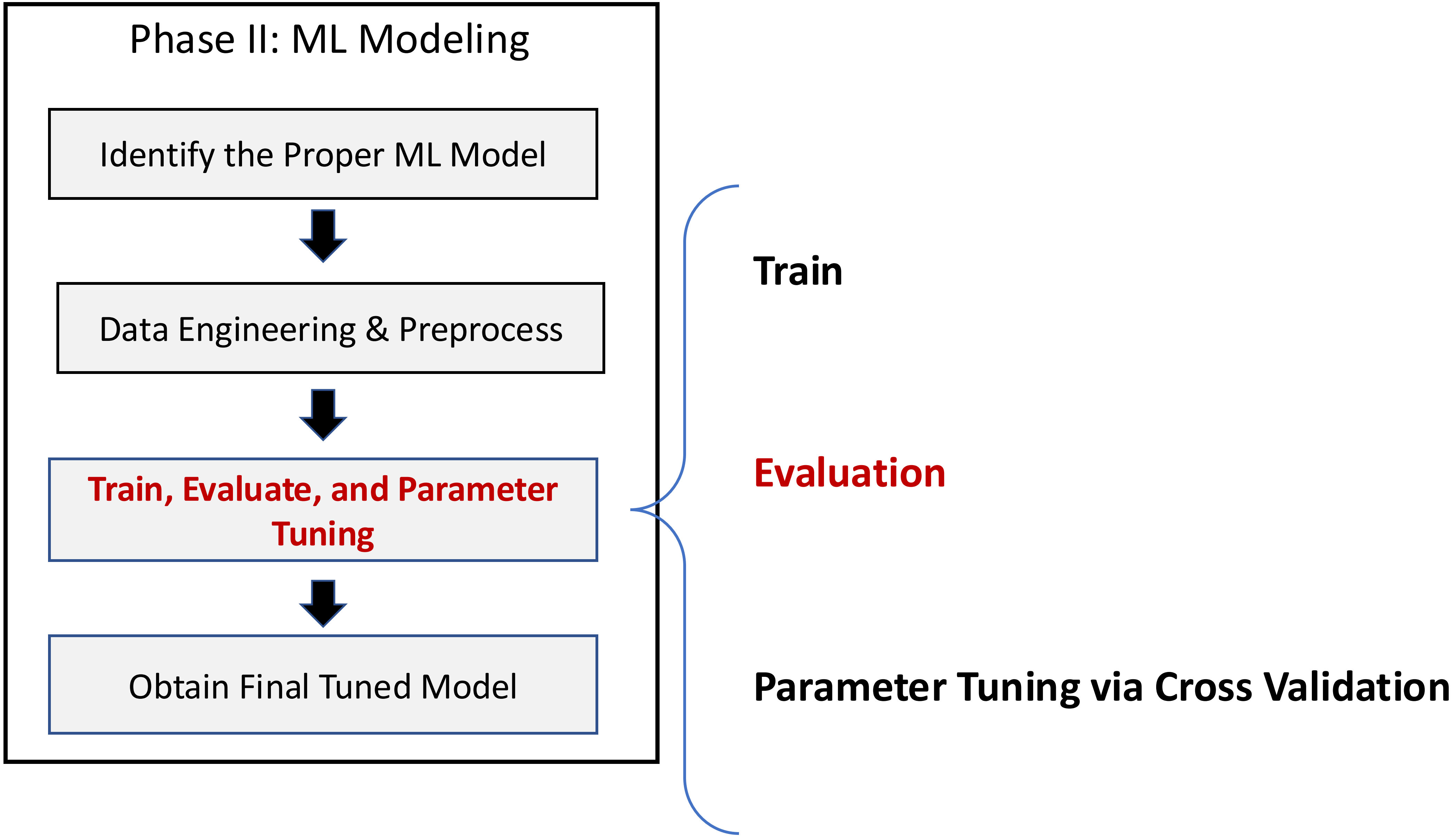
```
from pyspark.ml.classification import LogisticRegression

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

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

The fitted model is transformer

Philosophies of SparkML

- Everything is either an estimator or transformer
  - Feature engineering stages, like string indexer
  - ML stages, like logistic regression
  - Other steps in the whole process
- Pipeline connects estimators/transformers together, and pipeline itself is an estimator

- Benefits of this design: hide many low-level details, very convenient to use
- Drawbacks: difficult to customize low-level details
  - TensorFlow and PyTorch adopts very different philosophies (we will cover in future lectures)


Phase II: ML Modeling
Identify the Proper ML Model
Data Engineering & Preprocess
Train, Evaluate, and Parameter Tuning
Obtain Final Tuned Model
Train
Evaluation
Parameter Tuning via Cross Validation

Evaluation: Accuracy

Given a data set, the accuracy is defined as

$$Accuracy = \frac{\#\ of\ records\ correctly\ classified\ by\ our\ ML\ model}{total\ \#\ of\ records\ in\ the\ data\ set}$$

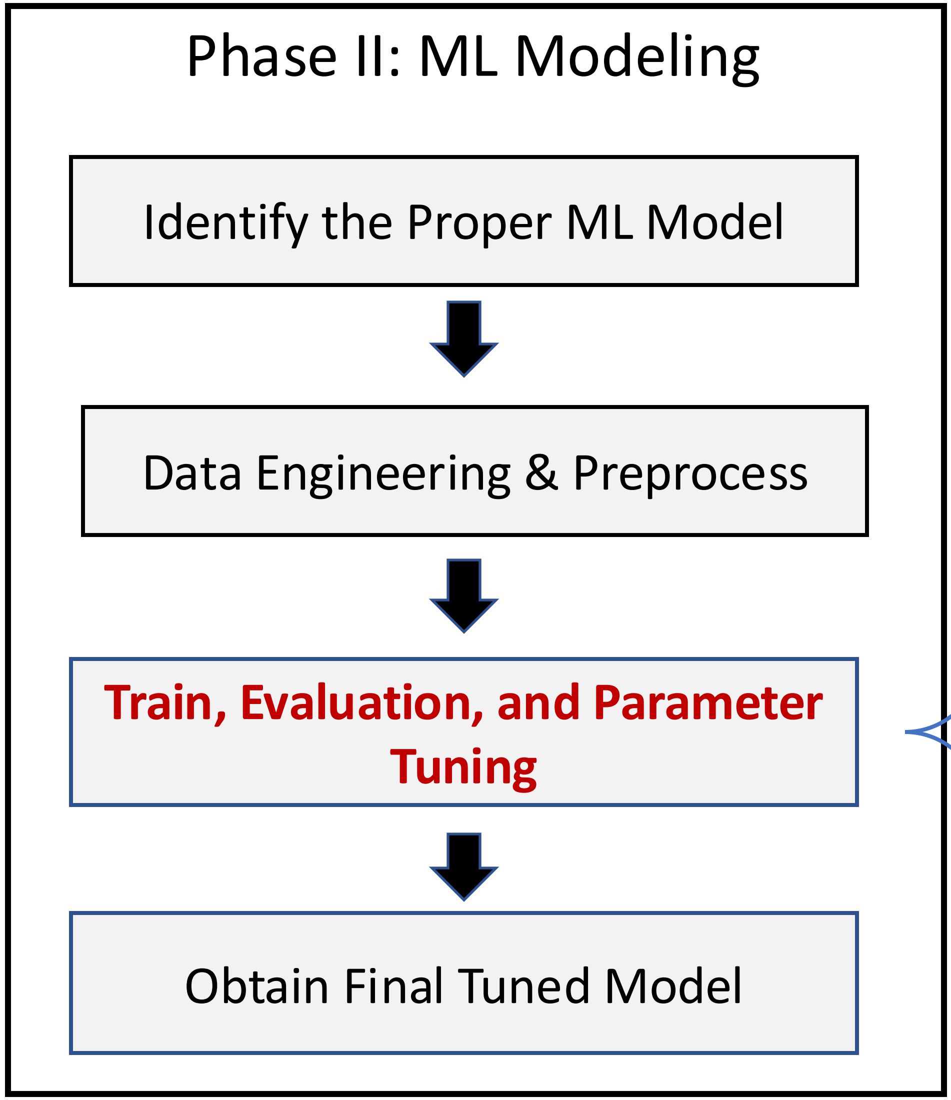
- Train accuracy: accuracy evaluated in training data set
- Test accuracy: accuracy evaluated in testing data set

Evaluation: Accuracy

```
predictions_train = lrModel.transform(nslkdd_df)# predictions using the training dataset
accuracy_train = (predictions_train.filter(predictions_train.outcome == predictions_train.prediction)
    .count() / float(predictions_train.count()))

accuracy_test = (predictions.filter(predictions.outcome == predictions.prediction)
    .count() / float(predictions.count()))
print(f"Train Accuracy : {np.round(accuracy_train*100,2)}%")
print(f"Test Accuracy : {np.round(accuracy_test*100,2)}%")
```

Get a subset of the rows of the dataframe where true outcome = prediction

```
Train Accuracy : 97.25%
Test Accuracy : 75.39%
```



Train

Evaluation

Other evaluation methodologies?

Parameter Tuning via Cross Validation

Evaluation: Confusion Matrix

Negative = normal
Positive = attack

True outcome (label):	Prediction: Negative	Prediction: Positive
Negative	True Negative	False Positive
Positive	False Negative	True Positive

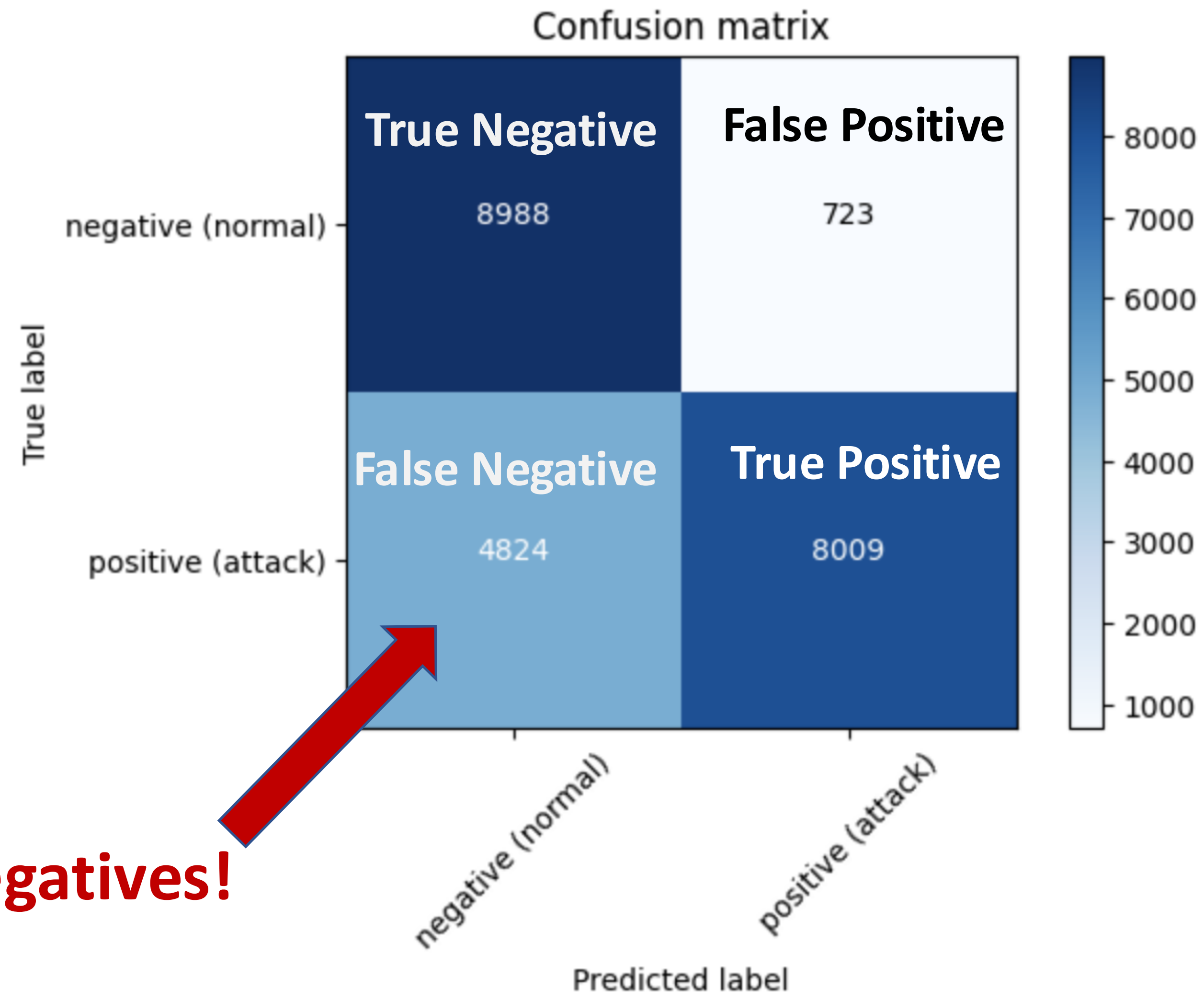
Evaluation: Confusion Matrix

```python
class_names=[0.0,1.0]
class_names_str=["negative (normal)","positive (attack)"]

outcome_true = predictions.select("outcome")
outcome_true = outcome_true.toPandas()

pred = predictions.select("prediction")
pred = pred.toPandas()

cnf_matrix = confusion_matrix(outcome_true, pred,labels=class_names)
#cnf_matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names_str,
                      title='Confusion matrix')
plt.show()
```

Getting outcome column and prediction column in pandas format

Getting the confusion matrix

Evaluation: Confusion Matrix



Too many false negatives!