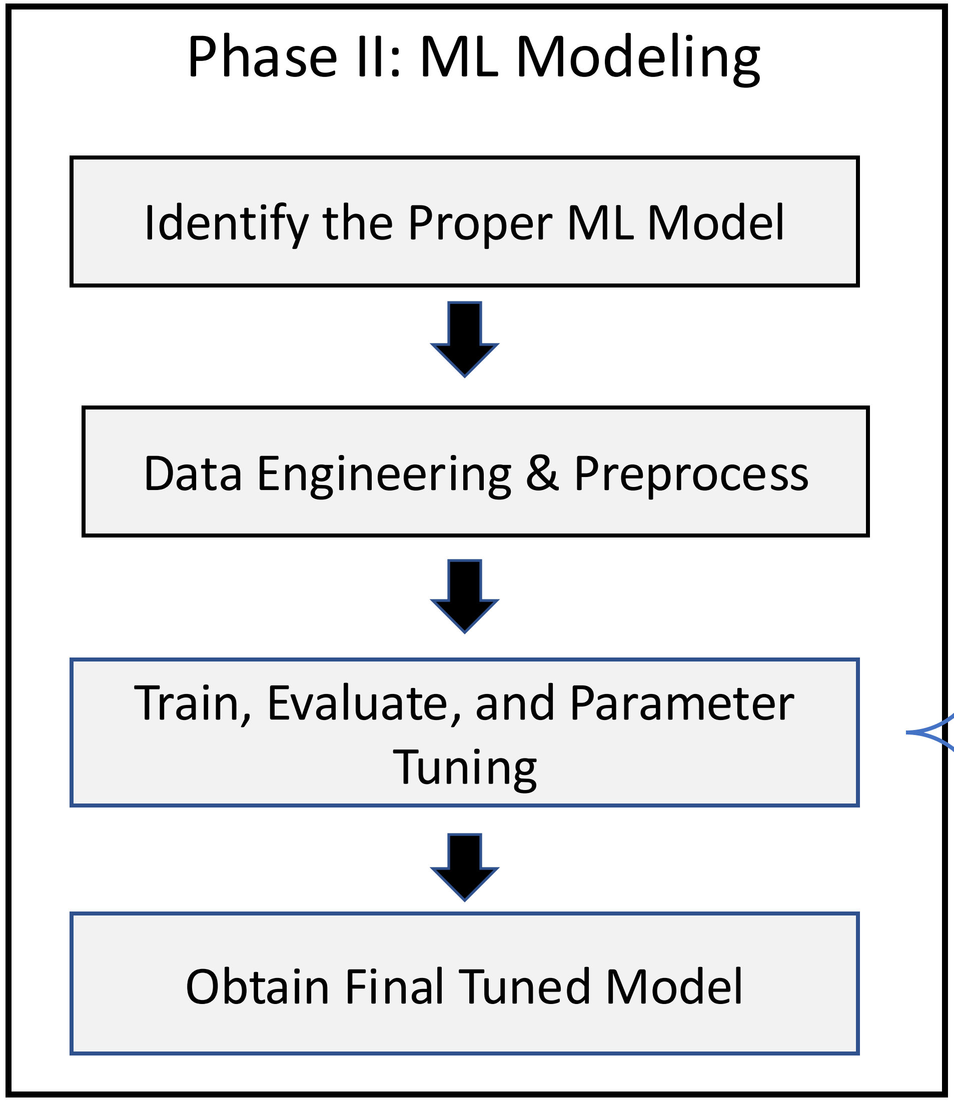
Machine Learning in Spark:
ML Models and Spark on the Cloud

Lecture 10 for 14-763/18-763

Guannan Qu

Oct 2, 2024


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

What we have covered...

Train

Evaluation
- Accuracy
- Confusion Matrix
- ROC/AUC

Hyper-Parameter Tuning via Cross Validation

Today’s Agenda

More ML Models:

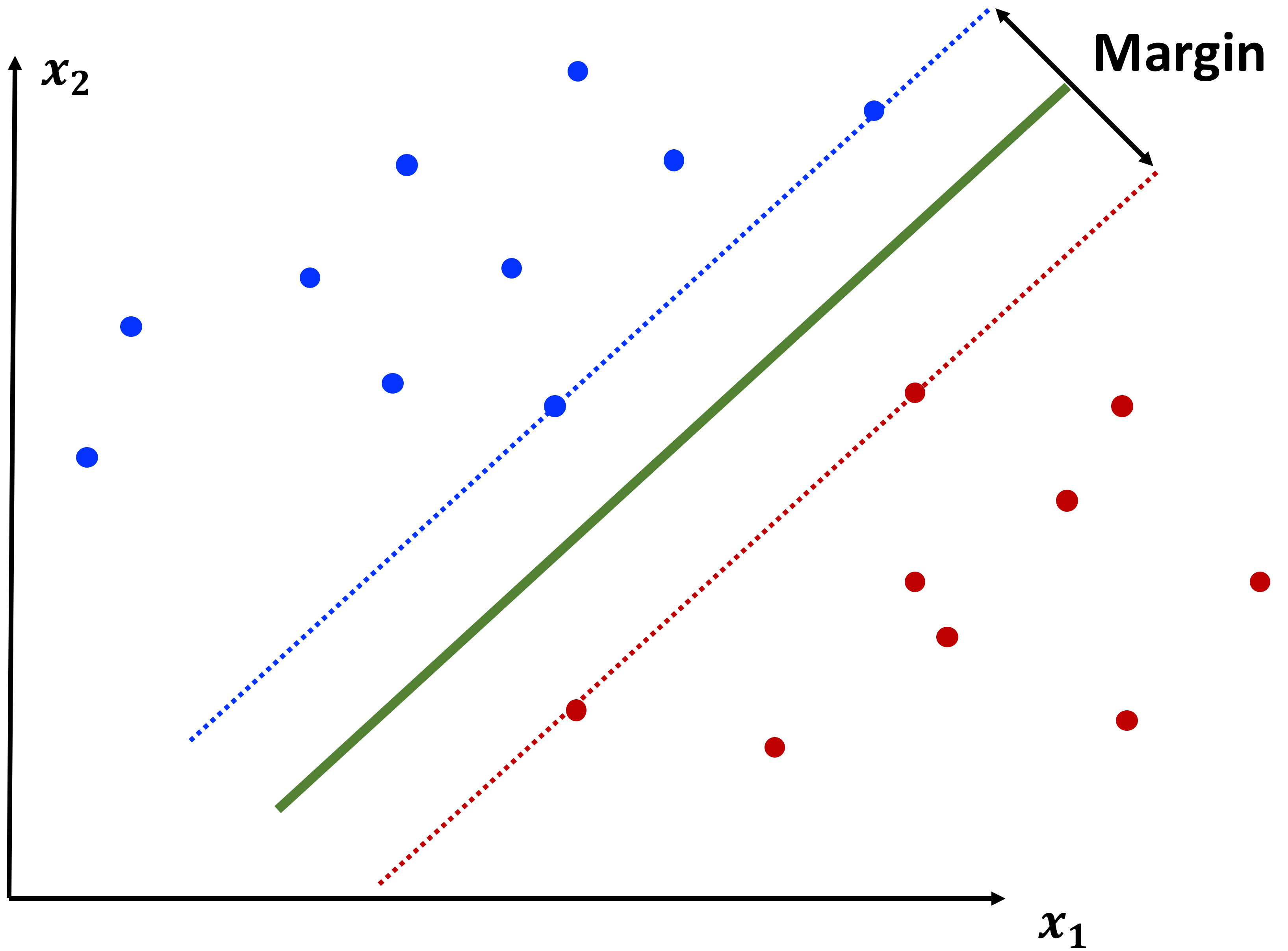
- Support Vector Machine
- Tree-like models
  - Decision tree
  - Random forest
- Keep in mind: this is not a theory ML course, we won’t go over the details of the model or the math equations. We will focus on:
  - “rough” idea on how it works
  - Pros and cons of each model
  - Key hyper-parameters, their role, and how to tune them.

Spark on the cloud

Support Vector Machine

- Proposed by Vladimir Vapnik and colleagues in AT&T Bell Laboratories in 1992
- It became popular due to success in handwritten digit recognition in 1994

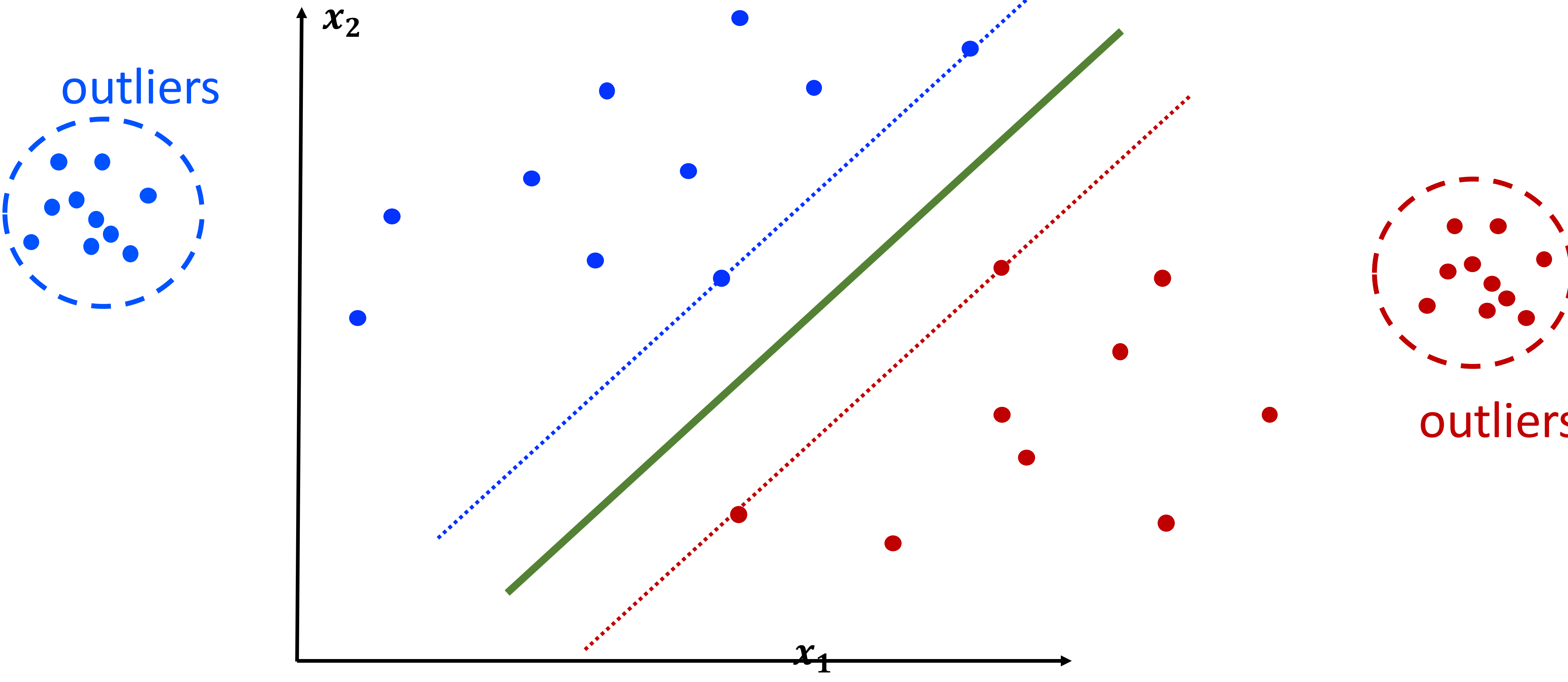
SVM finds a decision boundary with the largest margin!



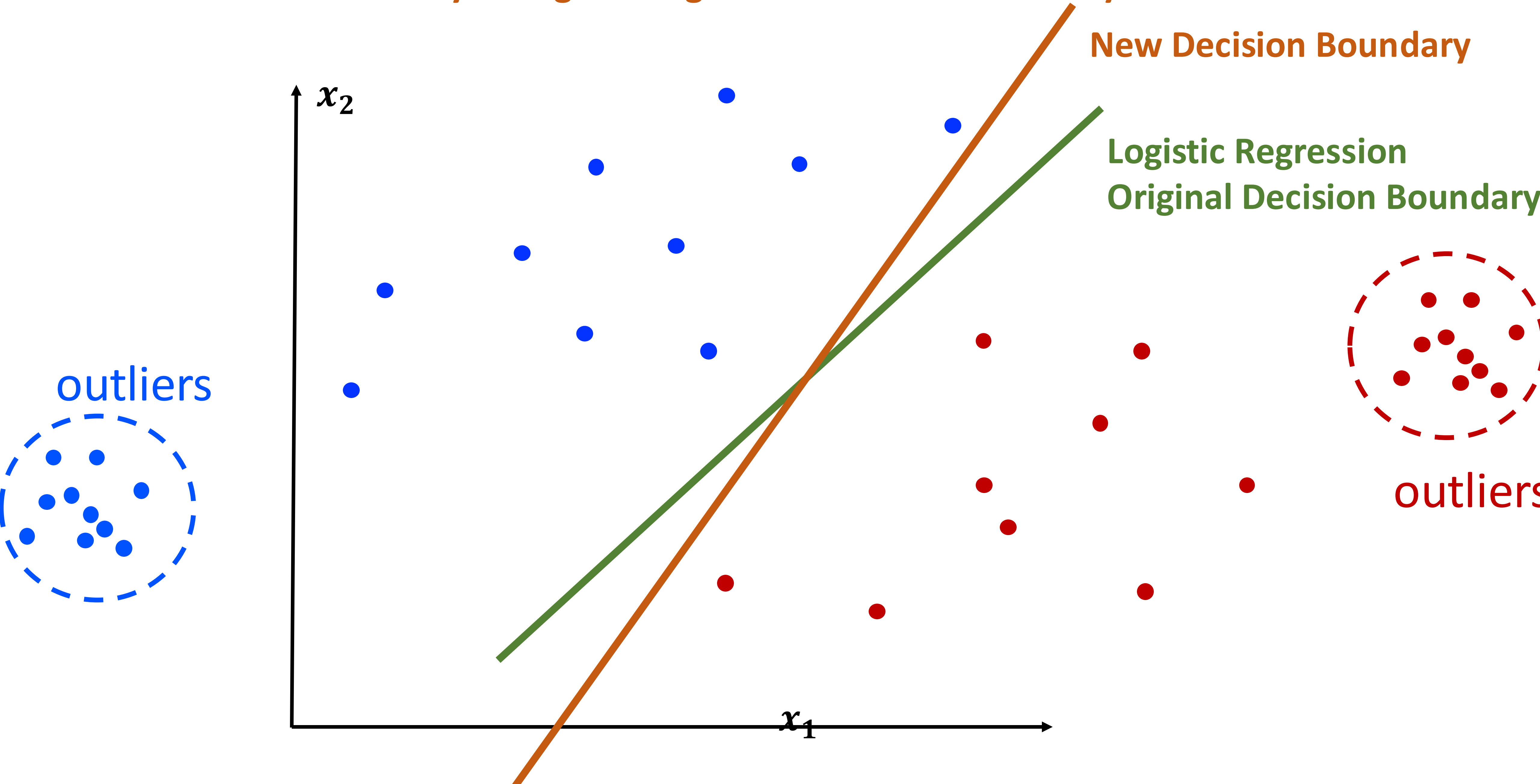
Support Vector Machine

- Pros
  - Less sensitive to (some) outliers

Decision Boundary of SVM is not affected by outliers!



Decision boundary of Logistic Regression will be affected by outliers!



Support Vector Machine

- Pros
    - Less sensitive to some outliers
    - Often uses less computation and memory as only “boundary” points are used to determine the decision boundary
- Cons
    - Do not provide a probability estimate
    - Works less well when the positive/negative points are not well separated
    - Similar to logistic regression, can only learn linear decision boundary

SVM in SparkML

```
from pyspark.ml.classification import LinearSVC

svm = LinearSVC(featuresCol="features",labelCol = "outcome")

svm_model = svm.fit(nslkdd_df)
```

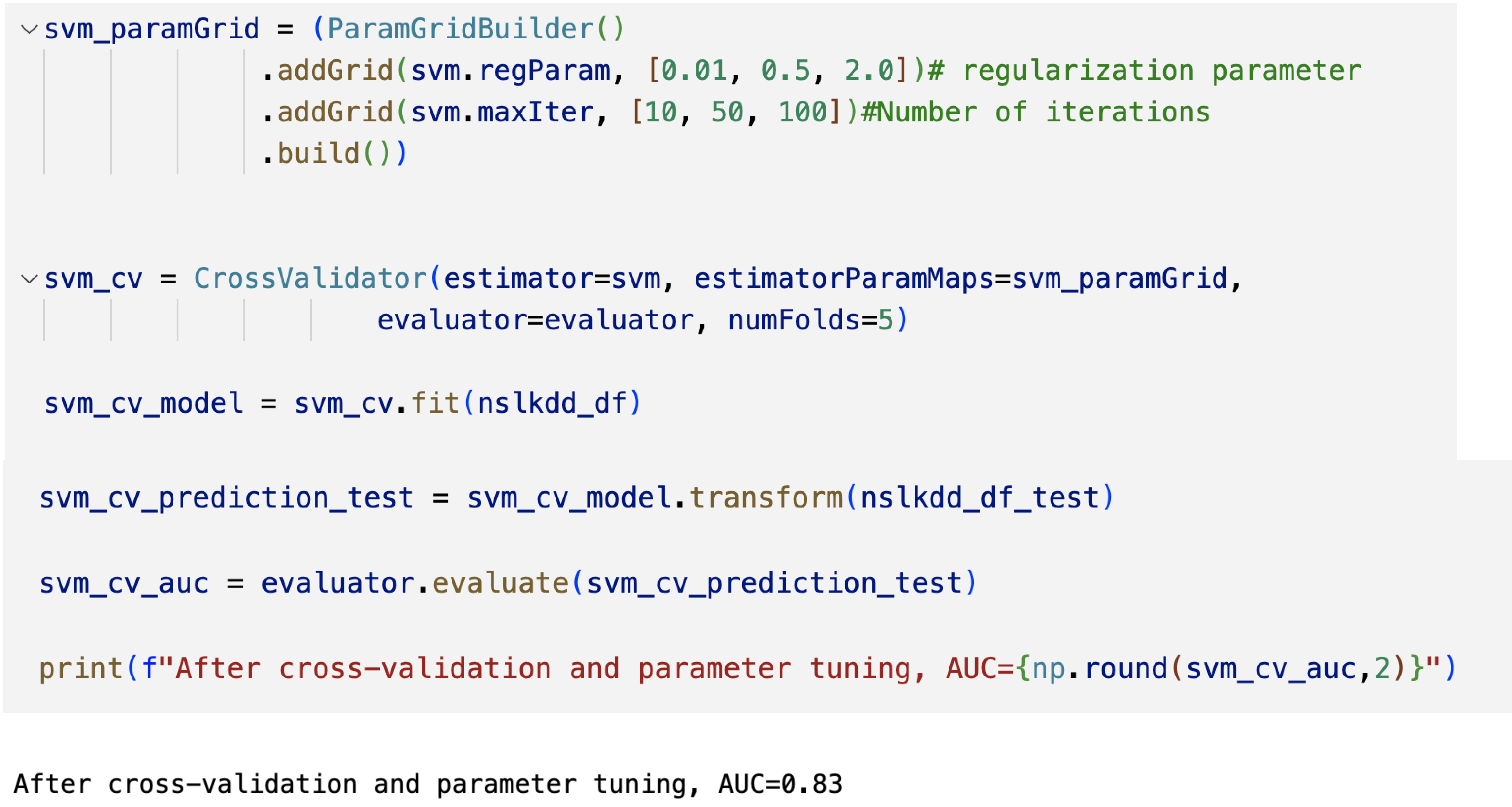
```python
# make predictions on training dataset and test data set
svm_prediction_train = svm_model.transform(nslkdd_df)
svm_prediction_test = svm_model.transform(nslkdd_df_test)

# calculate train and test accuracy
svm_accuracy_train = (svm_prediction_train.filter(
    svm_prediction_train.outcome == svm_prediction_train.prediction).count() /
    float(svm_prediction_train.count()))
svm_accuracy_test = (svm_prediction_test.filter(
    svm_prediction_test.outcome == svm_prediction_test.prediction).count()
    / float(svm_prediction_test.count()))

# calculate AUC
svm_auc = evaluator.evaluate(svm_prediction_test)

print(f"Train accuracy = {np.round(svm_accuracy_train*100,2)}%")
print(f"Test accuracy = {np.round(svm_accuracy_test*100,2)}%")
print(f"AUC = {np.round(svm_auc,2)}")
```

```
Train accuracy = 97.5%
Test accuracy = 75.39%
AUC = 0.82
```

Hyper-Parameter Tuning for SVM

- The training of SVM is similar to that of logistic regression
- Similarly, SVM has the **maxIter** and **regParam** that we can tune

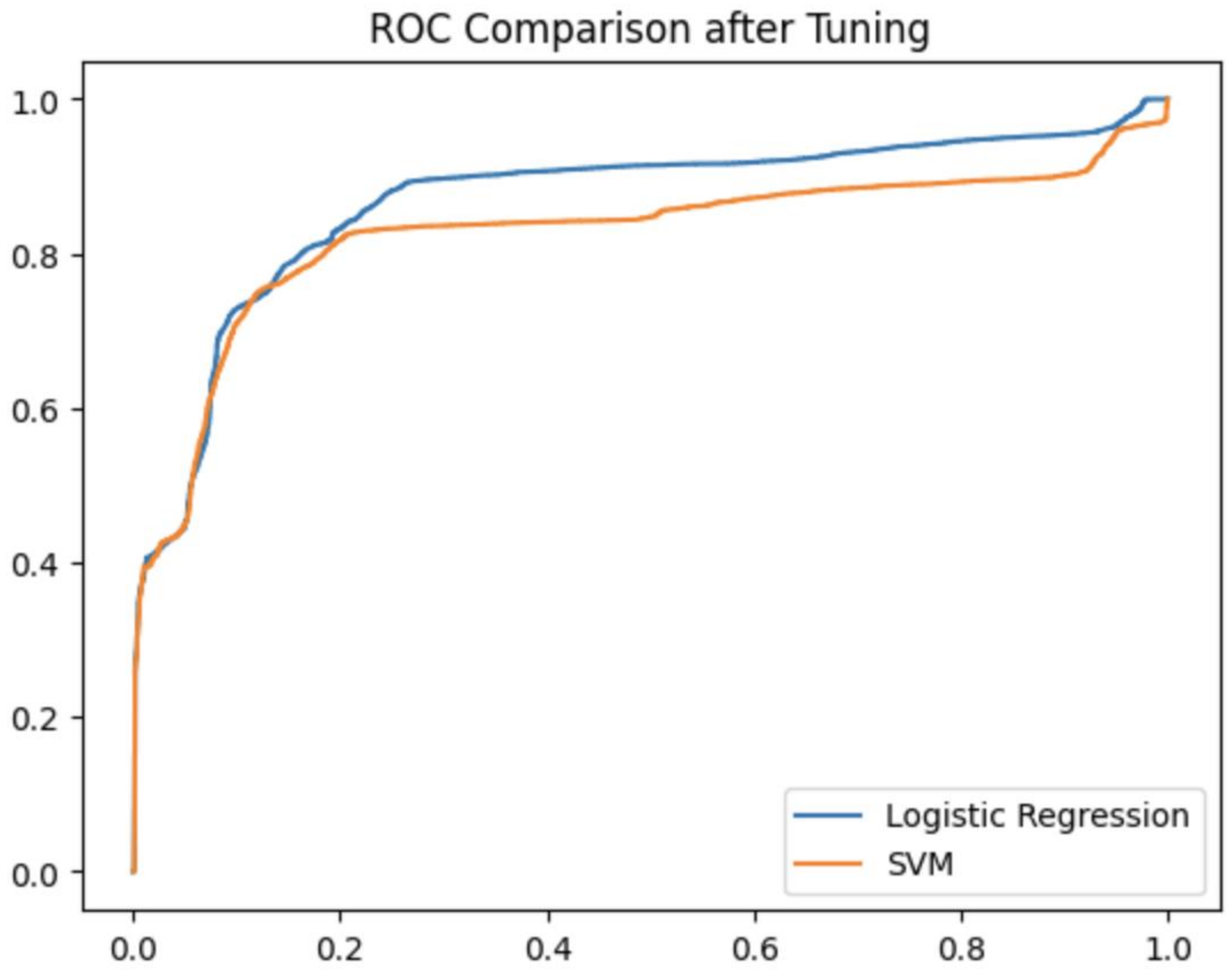
- **maxIter**: decides how many iterations we run the optimization solver
  - The larger the value, the higher precision we solve the minimization problem
  - If too large, does not improve the precision by much but can slow down fitting process
- **regParam**: controls the size of regularization
  -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



```
svm_paramGrid = (ParamGridBuilder()
                 .addGrid(svm.regParam, [0.01, 0.5, 2.0])# regularization parameter
                 .addGrid(svm.maxIter, [10, 50, 100])#Number of iterations
                 .build())


svm_cv = CrossValidator(estimator=svm, estimatorParamMaps=svm_paramGrid,
                        evaluator=evaluator, numFolds=5)

svm_cv_model = svm_cv.fit(nslkdd_df)

svm_cv_prediction_test = svm_cv_model.transform(nslkdd_df_test)

svm_cv_auc = evaluator.evaluate(svm_cv_prediction_test)

print(f"After cross-validation and parameter tuning, AUC={np.round(svm_cv_auc,2)}")
```

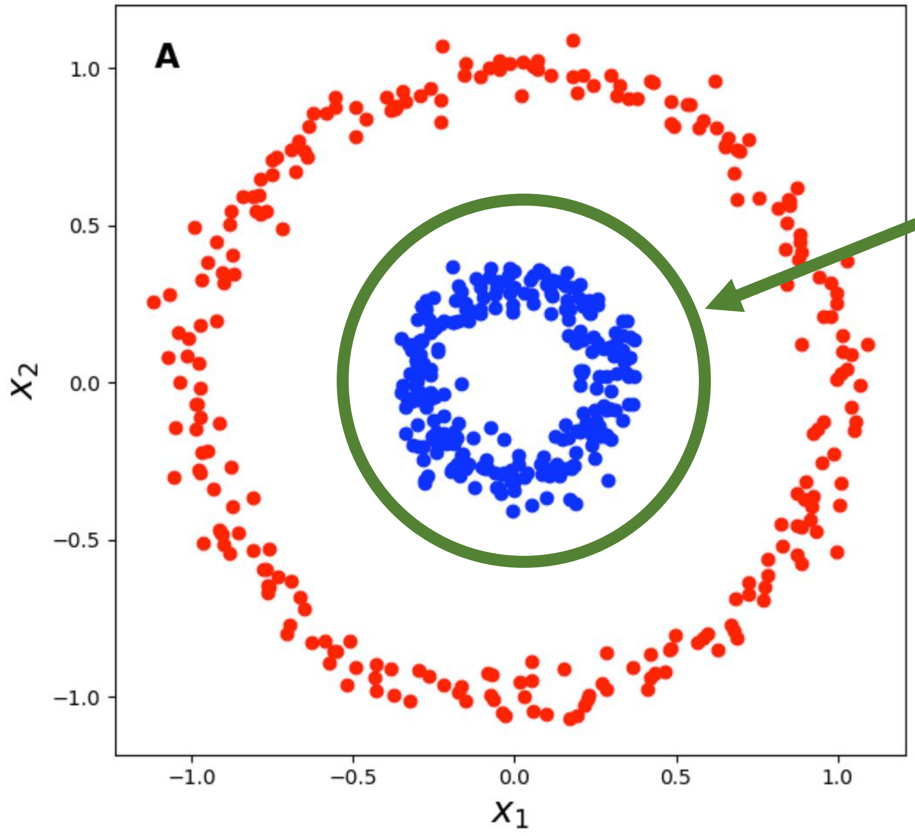
```
After cross-validation and parameter tuning, AUC=0.83
```

ROC Comparisons



Final Note for SVM

- SVM supports the “kernel trick”, which allows nonlinear decision boundaries



Decision Boundary of SVM with kernel trick

Optional Reading: https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

Common ML Models

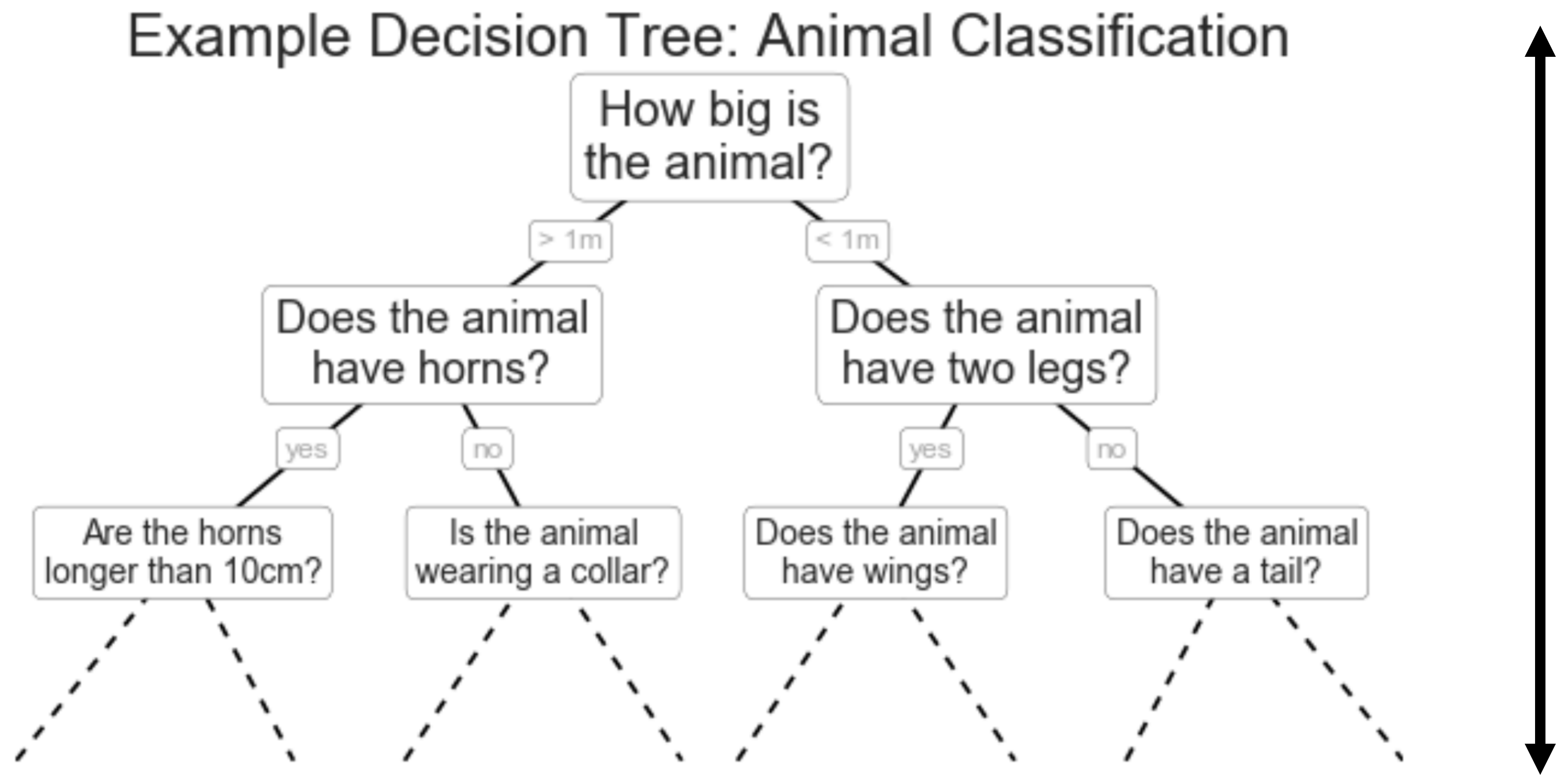
Name (Regressor/Classifier)	Representation Power	Tend to overfitting?	Pros/Cons	Hyper Parameters
Linear Regression/Logistic Regression	Linear	No	"Simplest" ML Model	maxIter/regParam
Support Vector Regression /Support Vector Machine	Linear	No	Pro: Less computation/memory, less sensitive to outliers Con: still "linear", doesn't work well if data not separated	maxIter/regParam
Kernel SVR/SVM	Limited nonlinearity	No	Same as SVR/SVM, but is a nonlinear model	Which kernel to use

Other ML Models

- Support Vector Machine

- Decision Tree
  - Underfitting vs overfitting
  - Random Forest

Decision Tree


Example Decision Tree: Animal Classification
How big is the animal?
> 1m
< 1m
Does the animal have horns?
Does the animal have two legs?
yes
no
yes
no
Are the horns longer than 10cm?
Is the animal wearing a collar?
Does the animal have wings?
Does the animal have a tail?

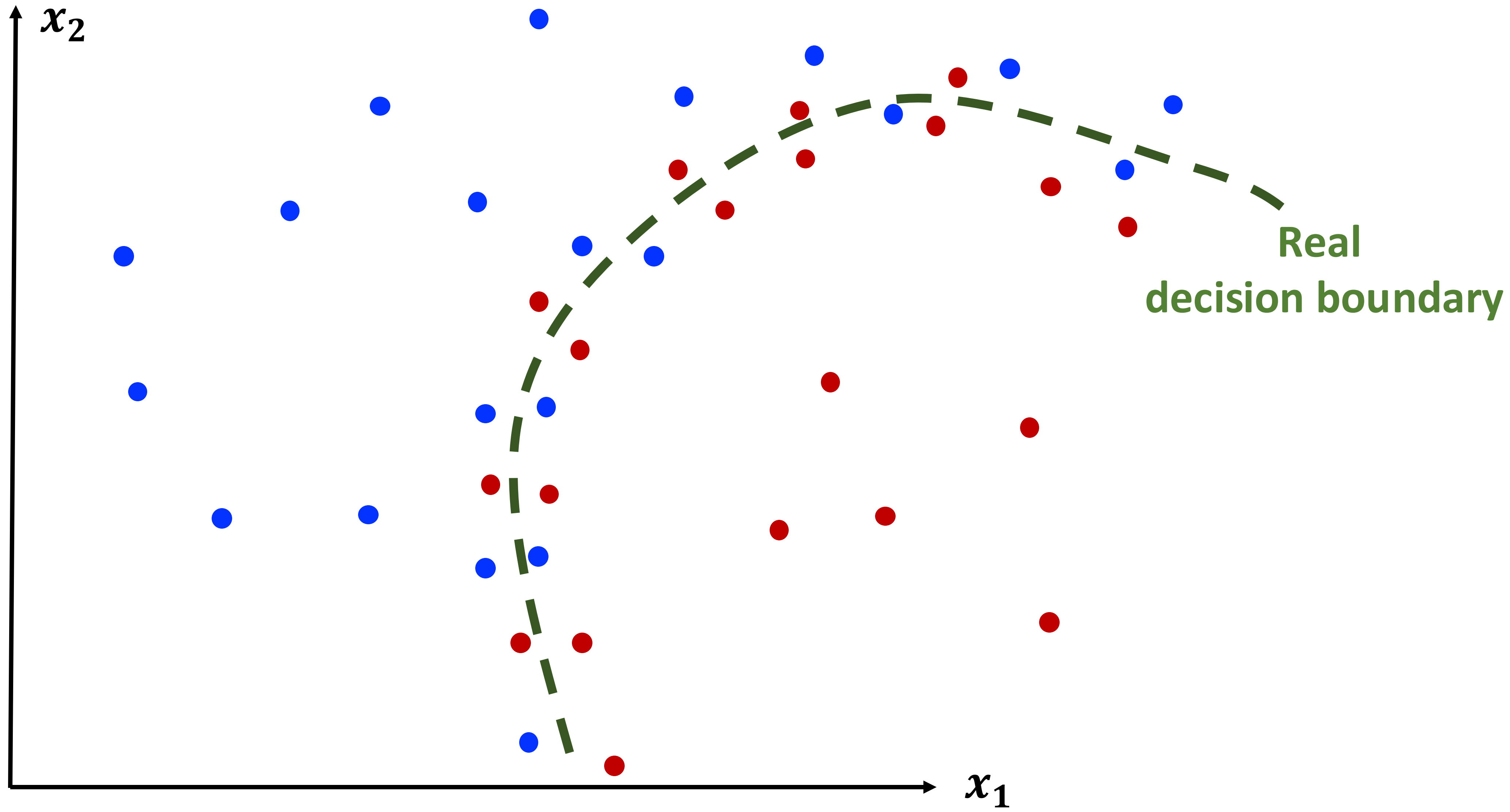
depth

Decision Tree

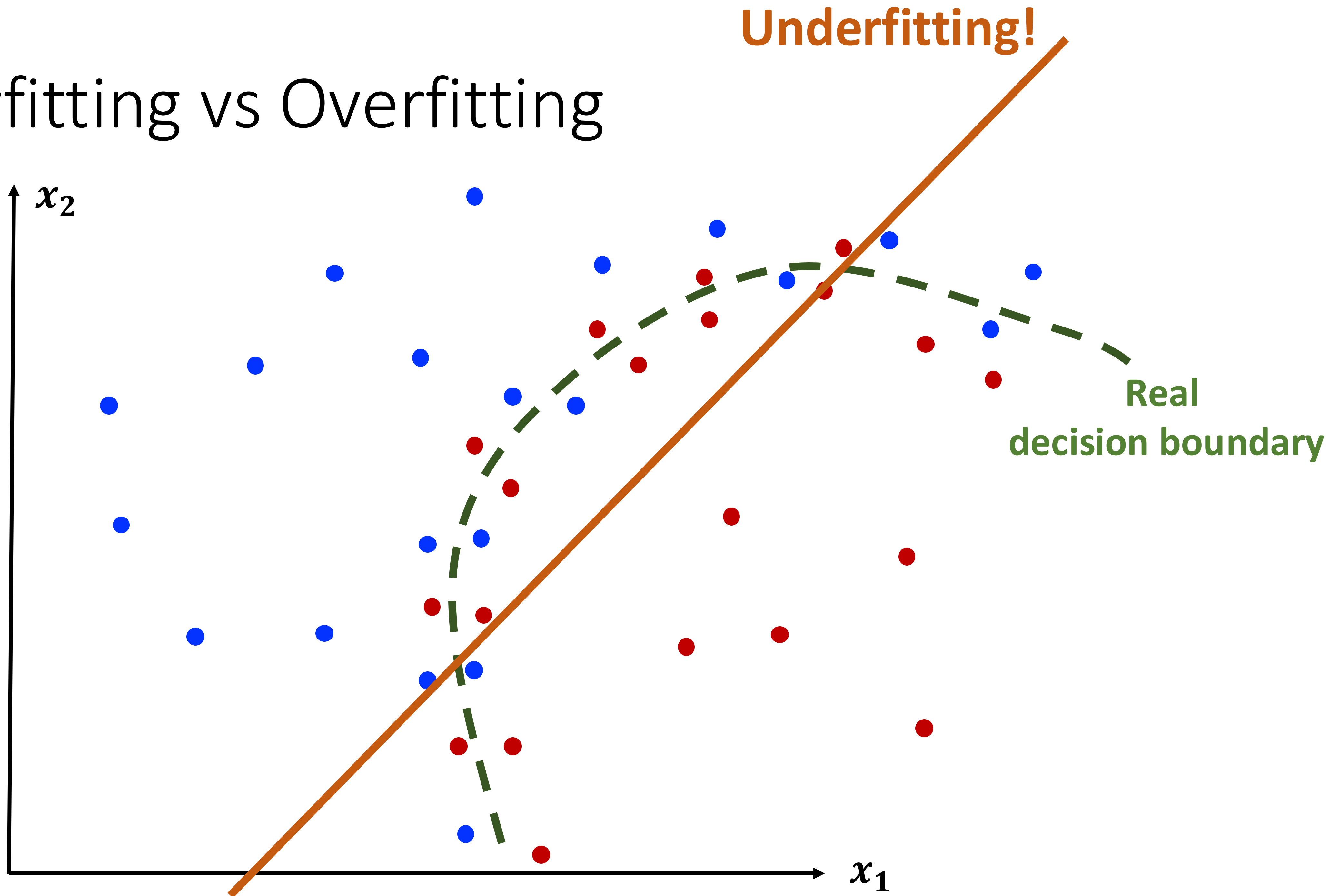
- Pros:
  - Explainable
  - Has a nonlinear decision boundary

- Cons:
  - Extremely prone to overfitting

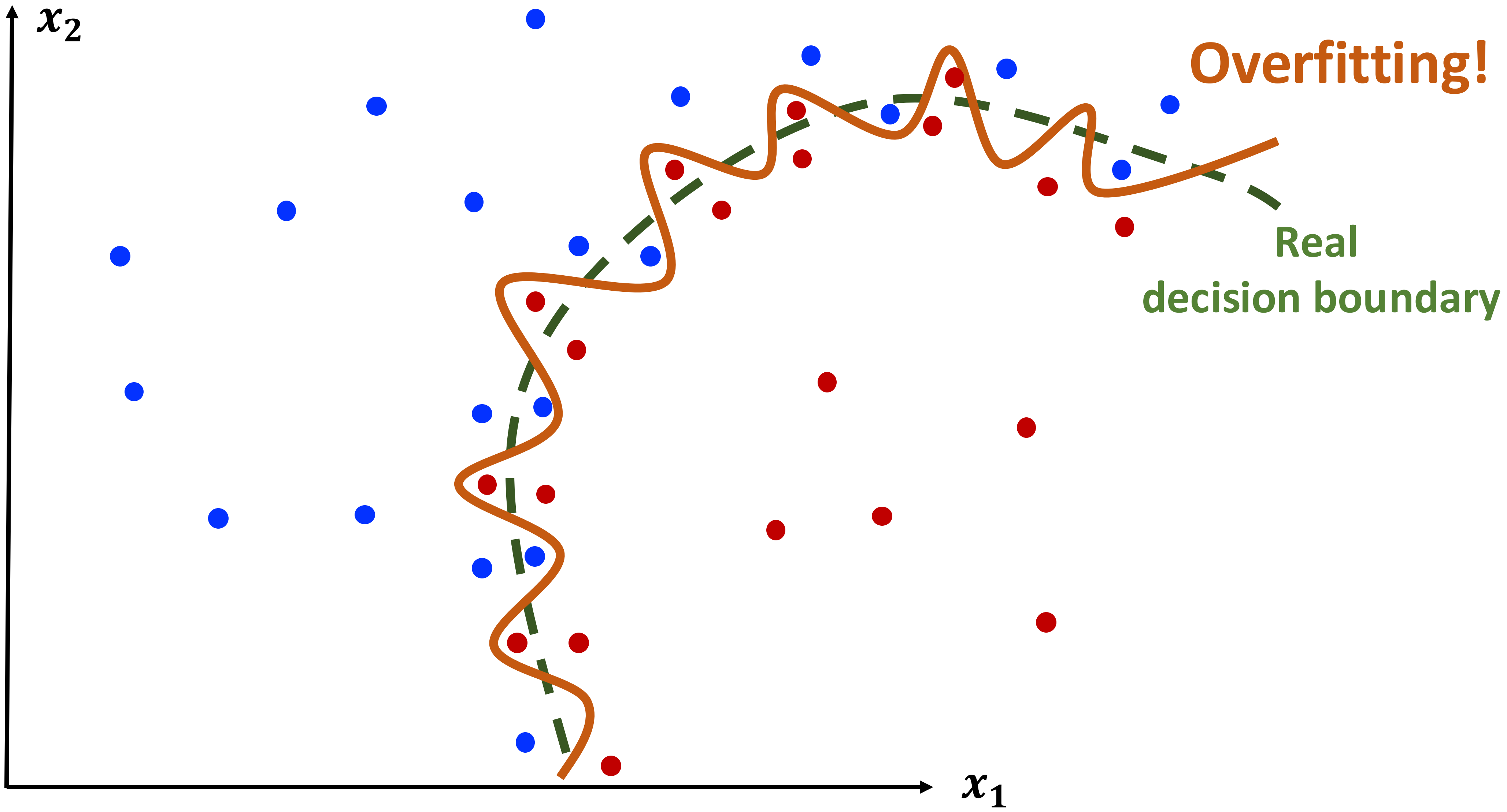
Underfitting vs Overfitting



Underfitting vs Overfitting



Underfitting vs Overfitting



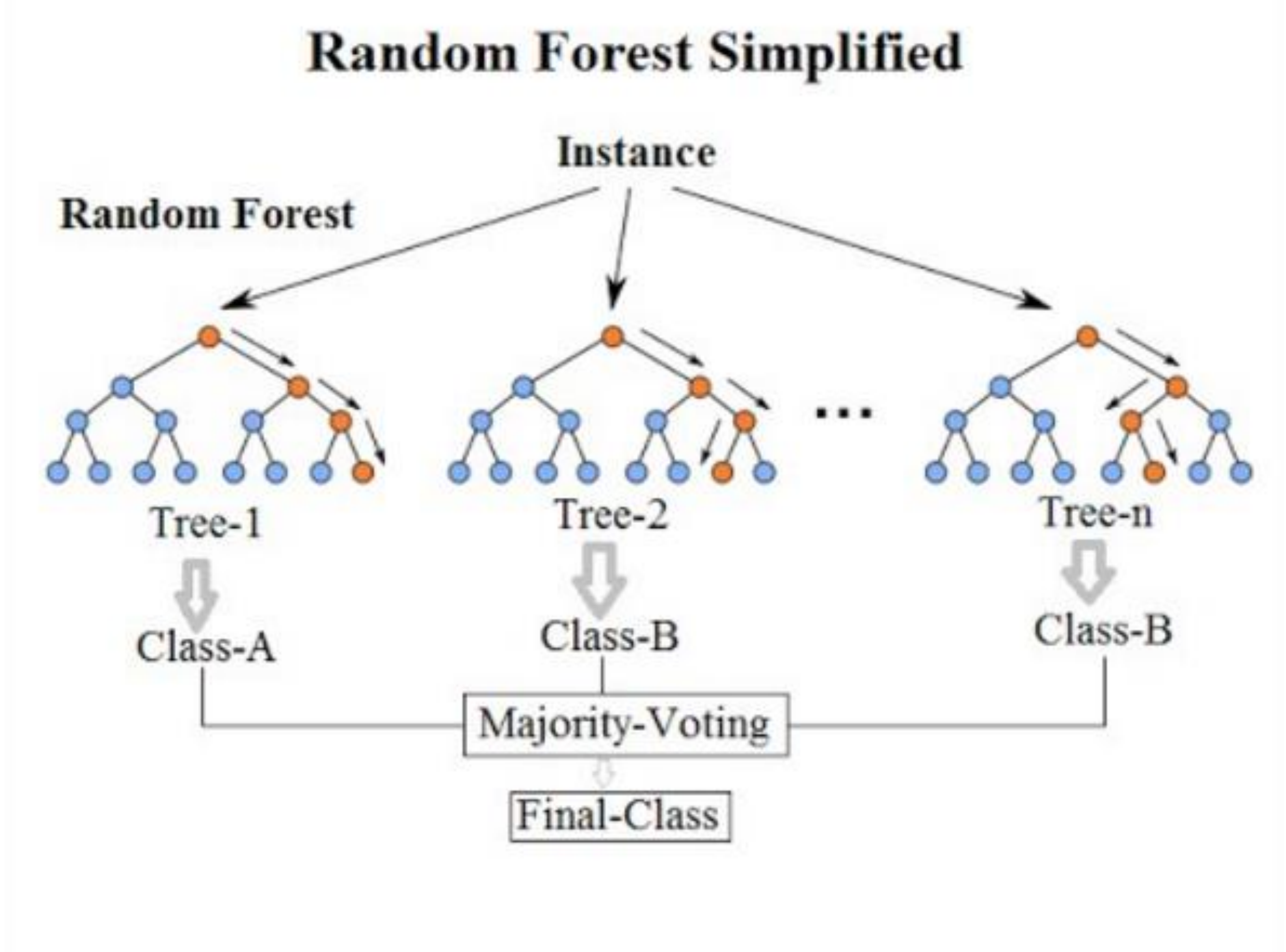
Underfitting vs Overfitting

- Underfitting:
  - The ML model is too simple to capture the real decision boundary
  - Example: linear decision boundary of logistic regression cannot learn complicated classification problems with nonlinear boundary

- Overfitting:
  - The ML model is too complex, and learns unnecessary “nuances”
  - Example: decision tree with unnecessarily large depths

Random Forest

How to counter overfitting of decision trees?
Random Forest use an ensemble of trees and aggregate their results to reduce overfitting!

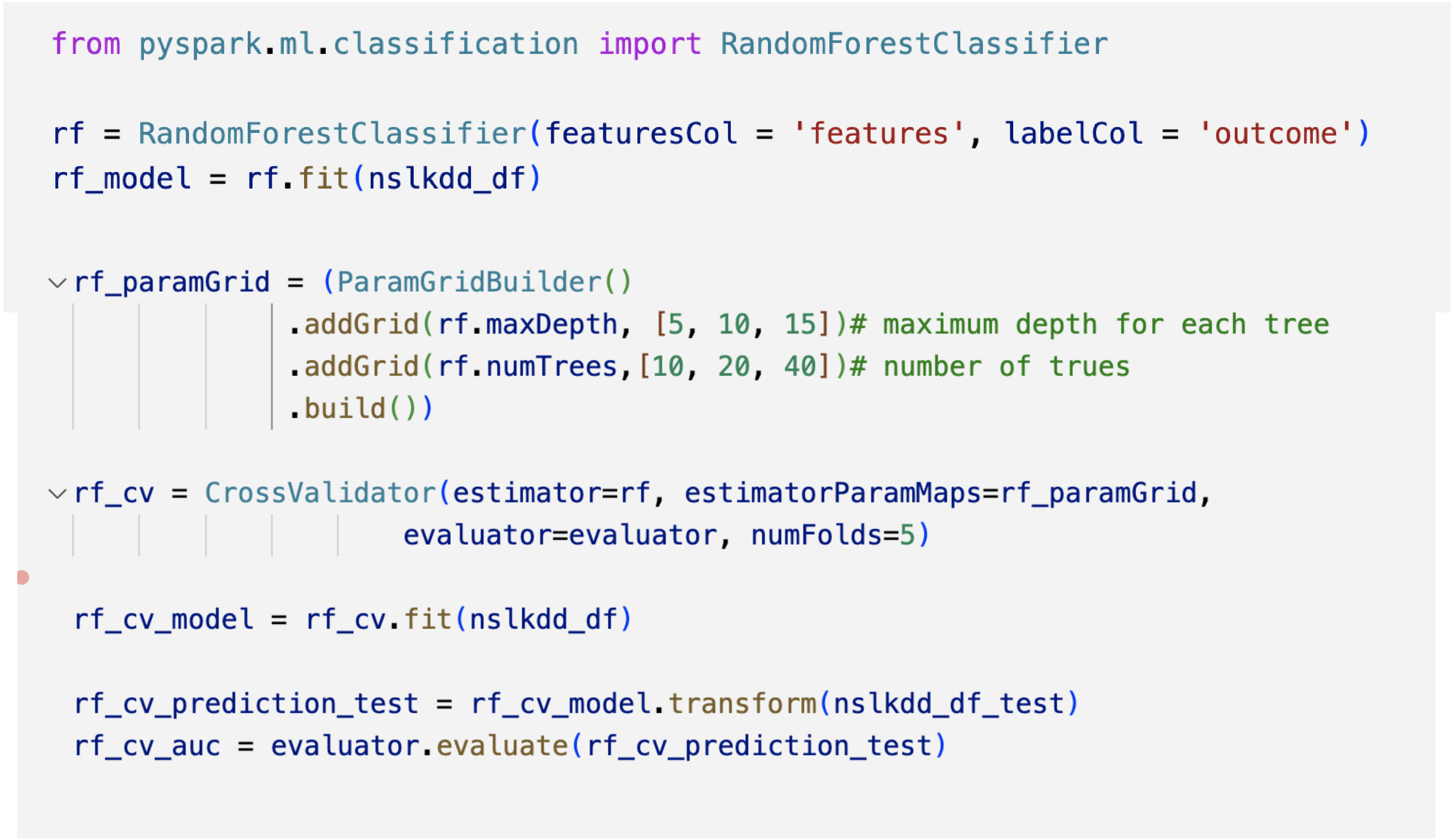


Random Forest

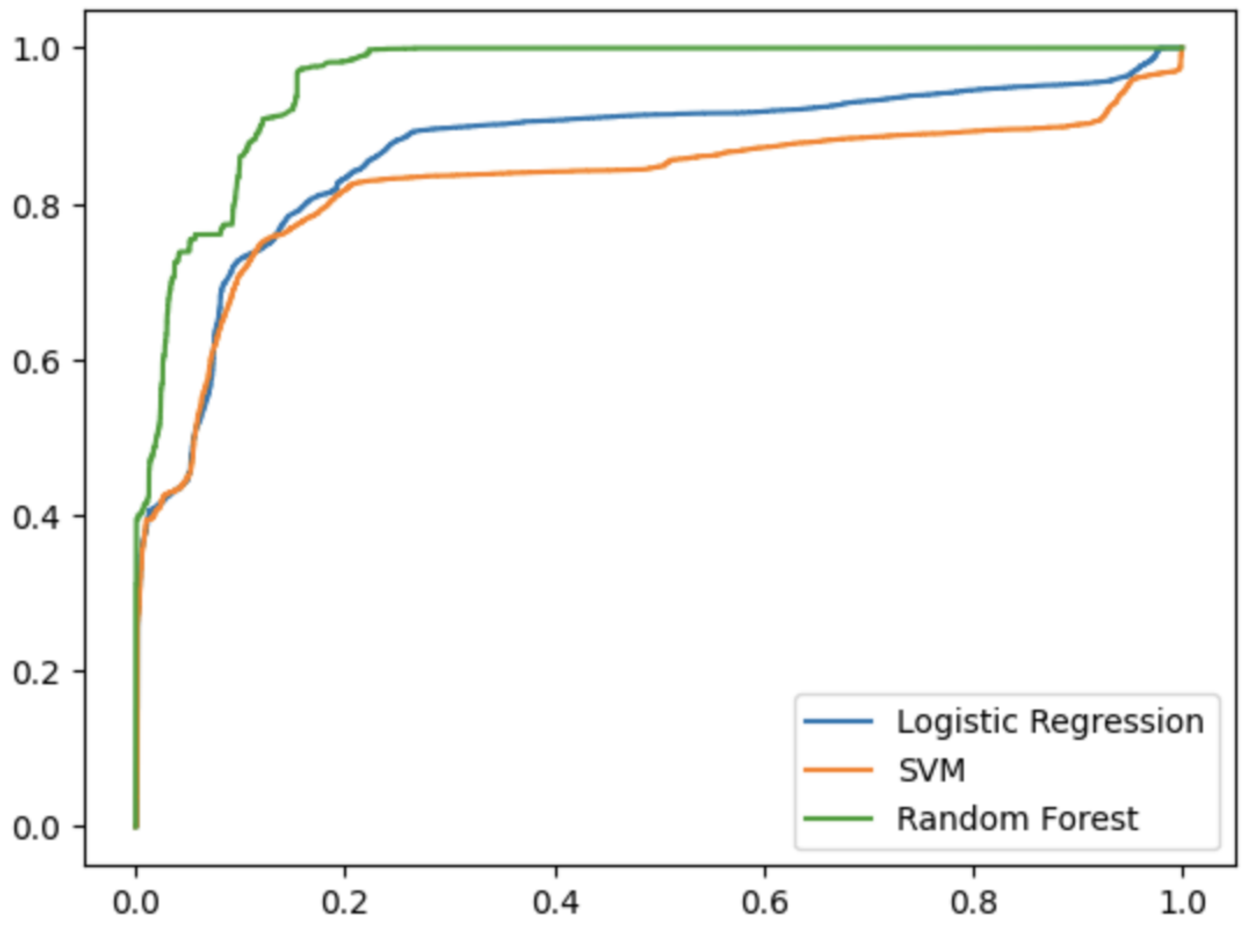
Key hyper-parameters

- numTrees: the number of trees.
  - Inrease numTrees will lead to less overfitting (thus better accuracy), but will slow down training
- maxDepth: the largest depth allowed for each tree.
  - Increase maxDepth will increase the ability to fit more complicated decision boundaries, but is more likely to overfit

- Pros: Can fit complex models, and doesn’t suffer from overfitting as in decision trees (if the hyper-parameters are chosen well)
- Cons: Can be slow to train as it fits many trees (but can be parallelized)



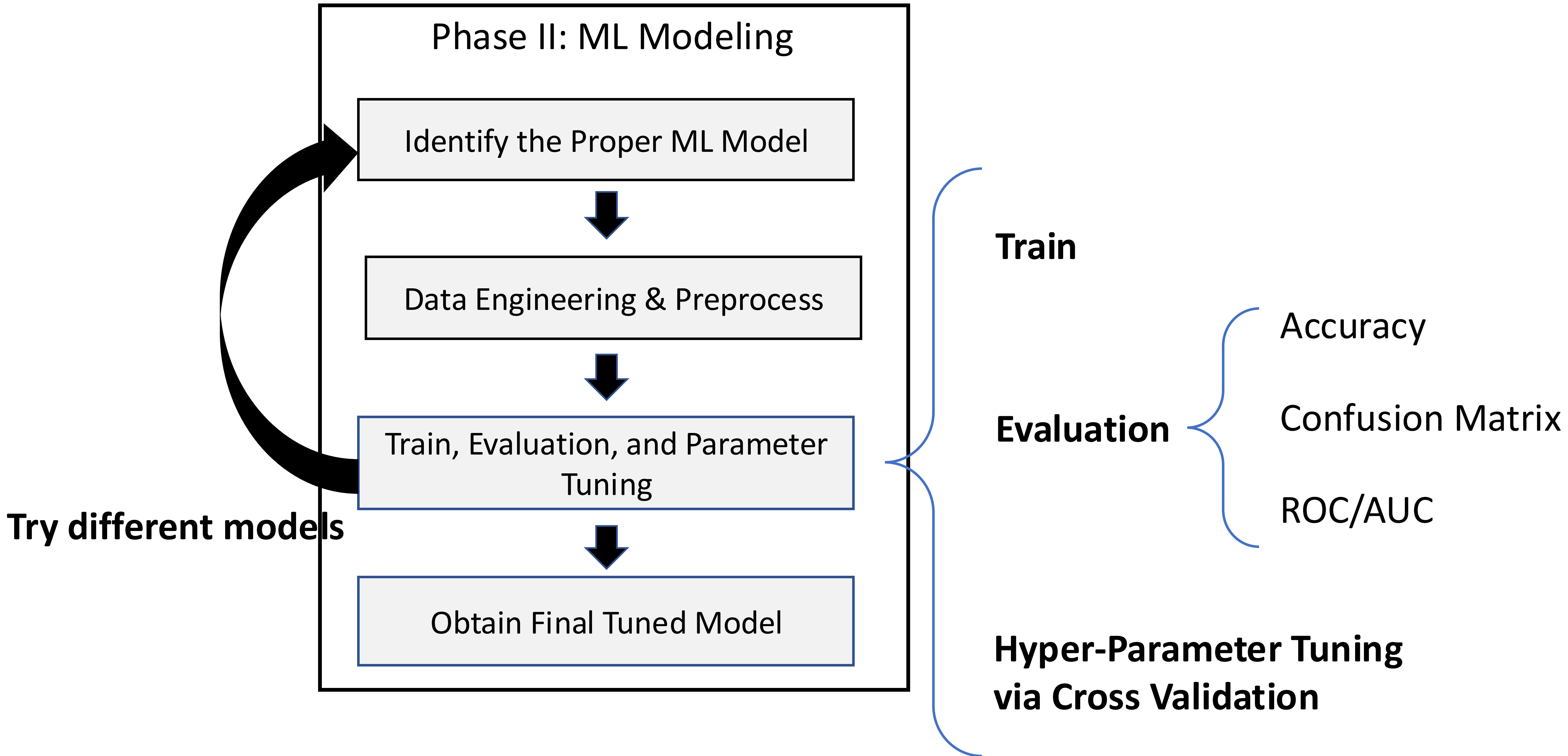
```python
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'outcome')
rf_model = rf.fit(nslkdd_df)


rf_paramGrid = (ParamGridBuilder()
            .addGrid(rf.maxDepth, [5, 10, 15])# maximum depth for each tree
            .addGrid(rf.numTrees,[10, 20, 40])# number of trues
            .build())

rf_cv = CrossValidator(estimator=rf, estimatorParamMaps=rf_paramGrid,
                    evaluator=evaluator, numFolds=5)

rf_cv_model = rf_cv.fit(nslkdd_df)

rf_cv_prediction_test = rf_cv_model.transform(nslkdd_df_test)
rf_cv_auc = evaluator.evaluate(rf_cv_prediction_test)
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression
SVM
Random Forest

Common ML Models

Name (Regressor/Classifier)	Representation Power	Tend to overfitting?	Pros/Cons	Hyper Parameters
Linear Regression/Logistic Regression	Linear	No	"Simplest" ML Model	maxIter/regParam
Support Vector Regression /Support Vector Machine	Linear	No	Pro: Less computation/memory, less sensitive to outliers Con: still "linear", doesn't work well if data not separated	maxIter/regParam
Kernel SVR/SVM	Limited nonlinearity	No	Same as SVR/SVM, but is a nonlinear model	Which kernel to use
Random Forests	Nonlinear	No (if well-tuned)	Pros: fits complex models, with not so much overfitting Cons: computationally heavy	numTrees/maxDepth

Summary for SparkML

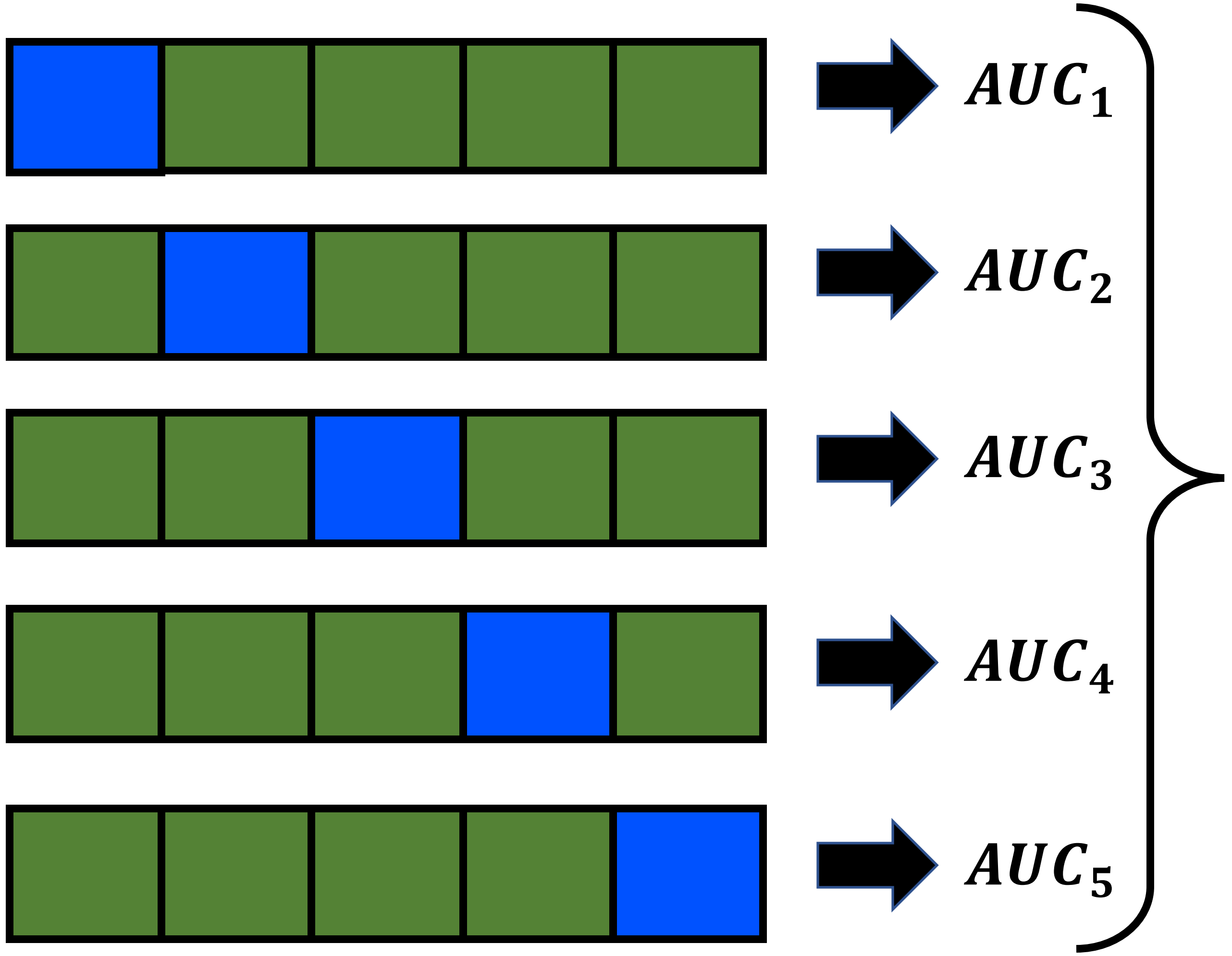

Phase II: ML Modeling
Identify the Proper ML Model
Data Engineering & Preprocess
Train, Evaluation, and Parameter Tuning
Obtain Final Tuned Model
Try different models
Train
Evaluation
Accuracy
Confusion Matrix
ROC/AUC
Hyper-Parameter Tuning via Cross Validation

SparkML distributed computing

- Syntax-wise, sparkML is very similar to scikit-learn (an extremely popular python ML package). If you know how to use spark, you will also know how to use scikit-learn

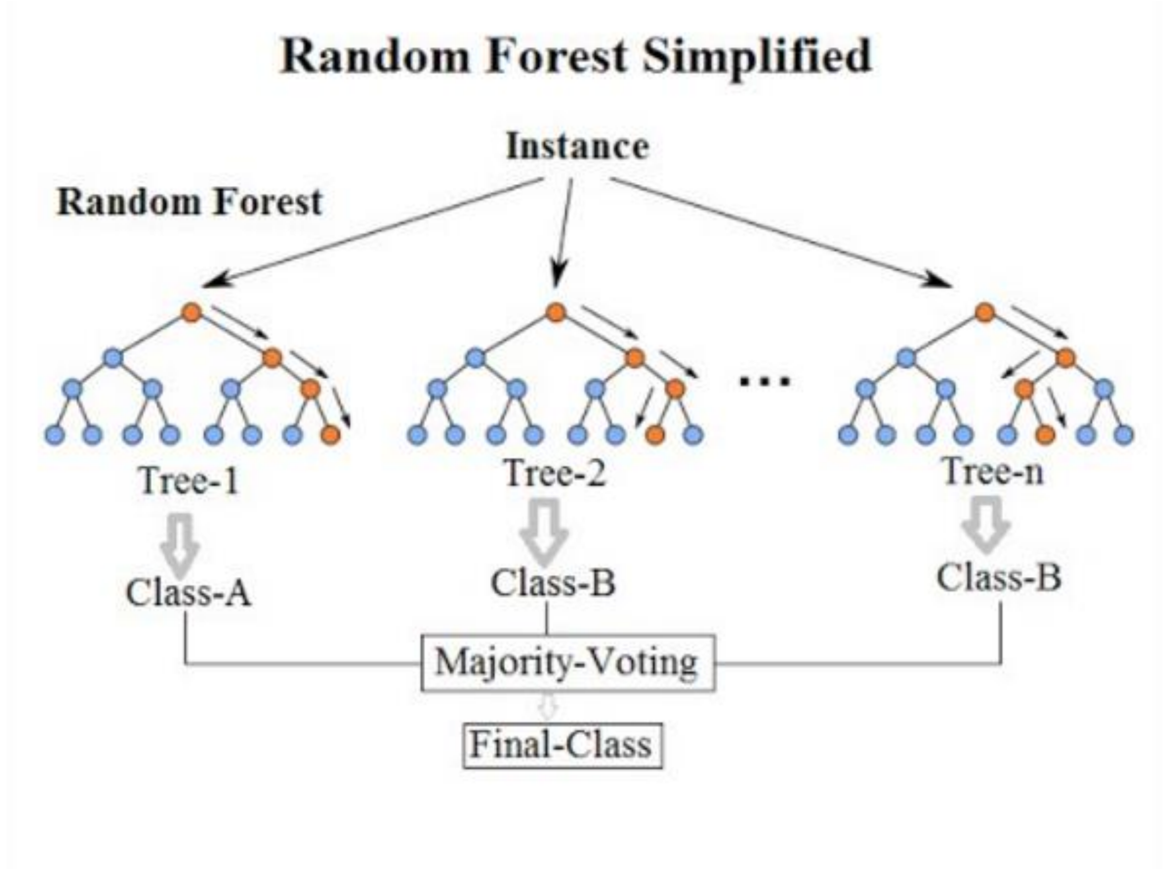
- What makes sparkML different is that it is a “distributed” data processing platform that can runs on a variety of settings, including on local machines or a cluster of multiple workers

SparkML distributed computing



The cross-validation process can be parallelized!

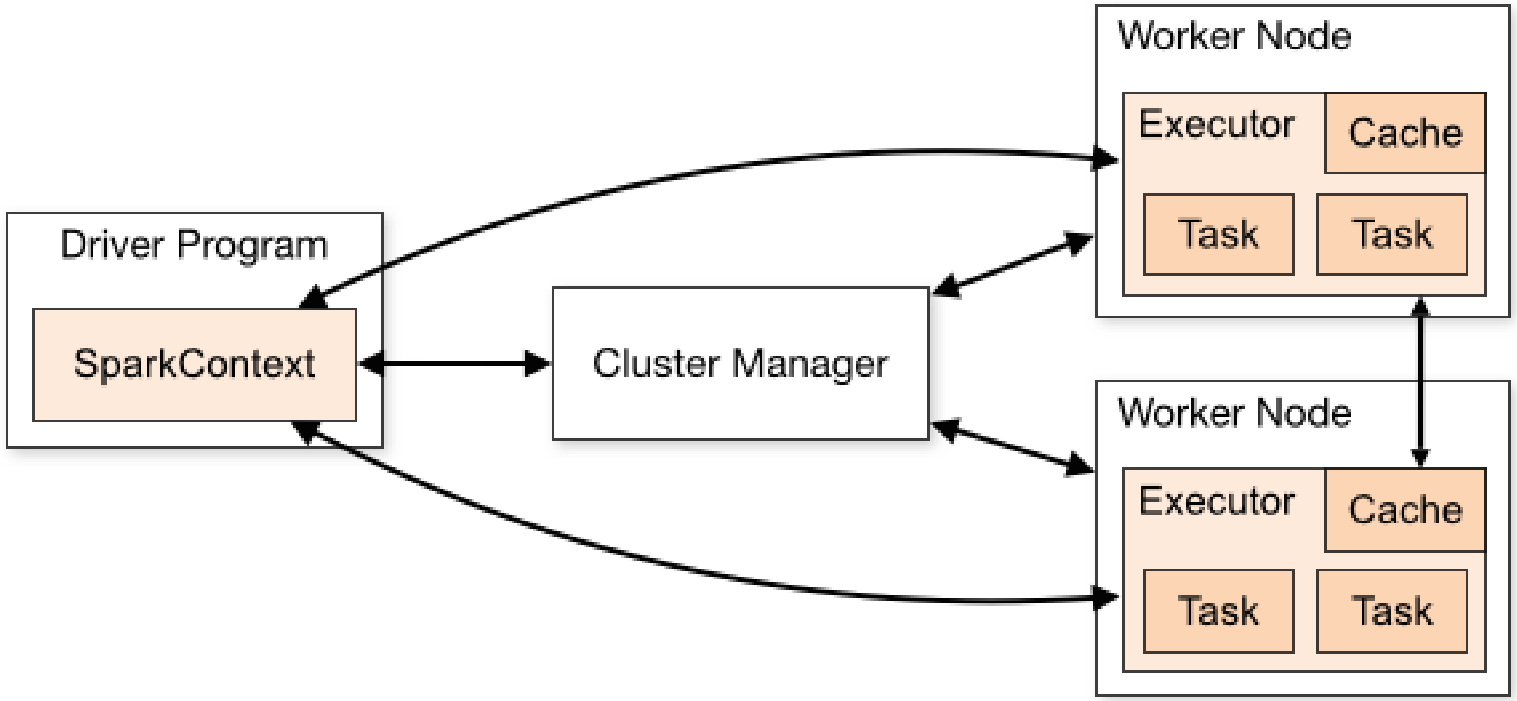
SparkML distributed computing



Random Forest Training can also be parallelized!

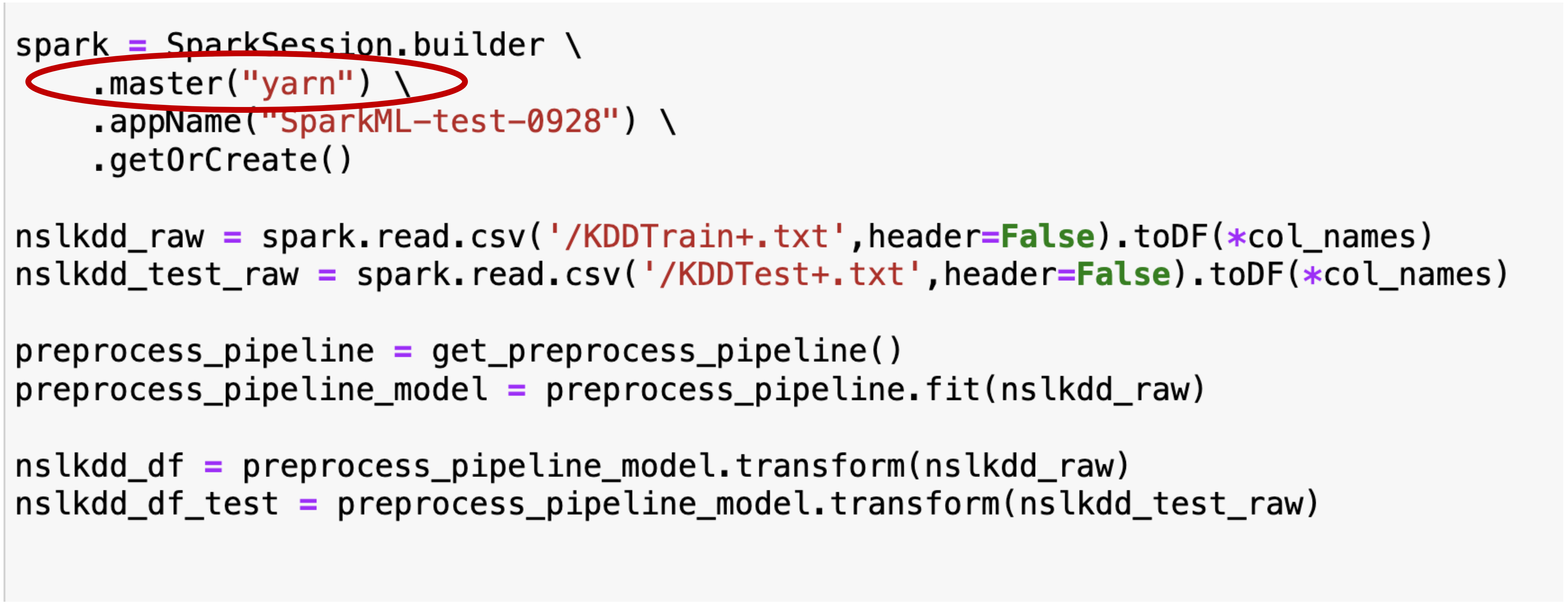
Lab: SparkML on the cloud

- Goal: run SparkML on a cluster on the cloud



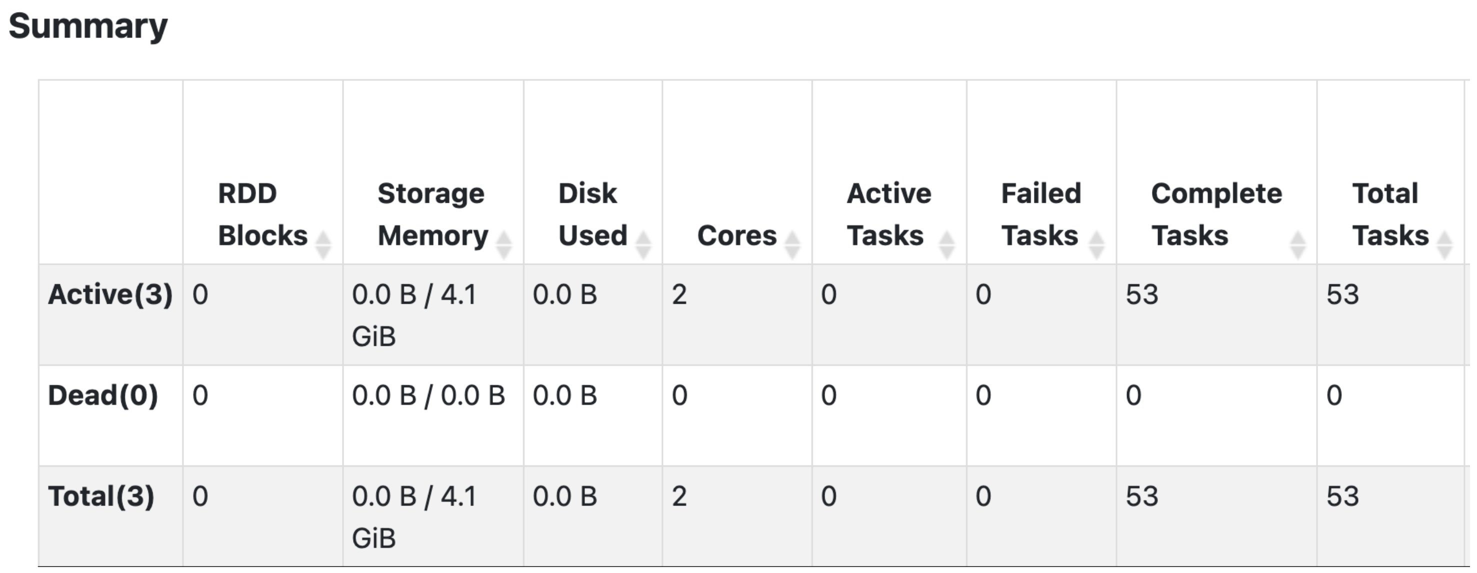
Lab: SparkML on the cloud

- On dataproc cluster, when creating the Spark Session, set master as "yarn".



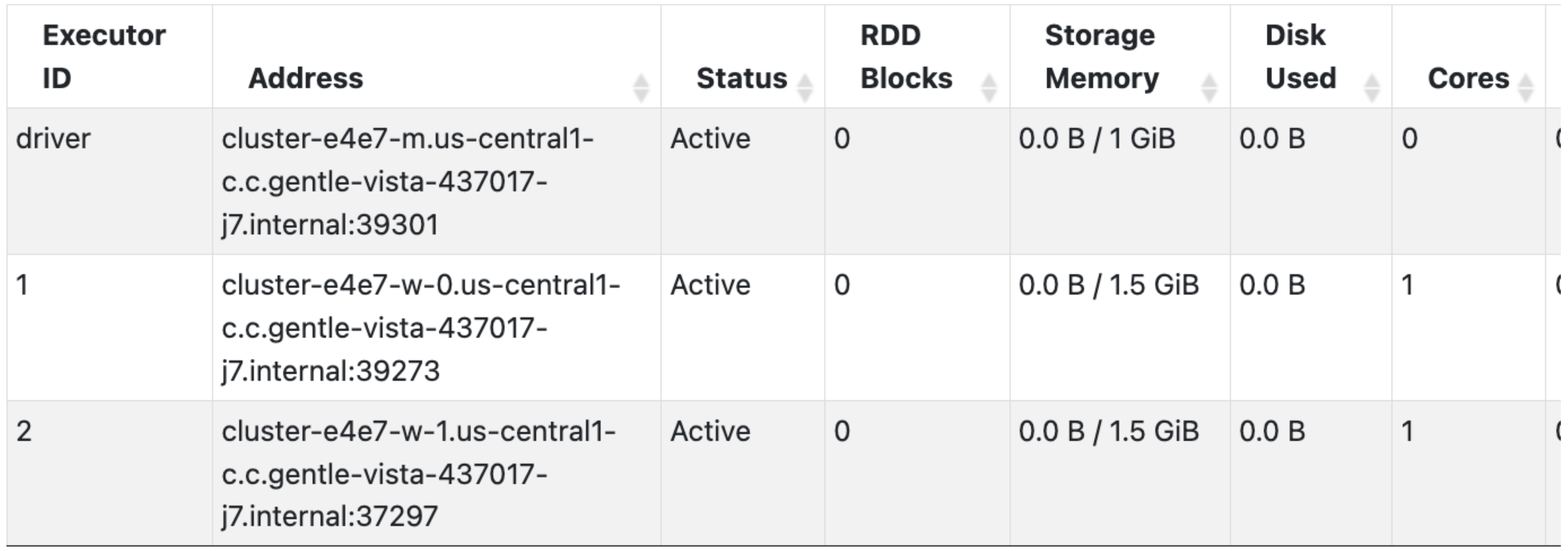
```
spark = SparkSession.builder \
    .master("yarn") \
    .appName("SparkML-test-0928") \
    .getOrCreate()

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

Execution Breakdown in Cluster Mode

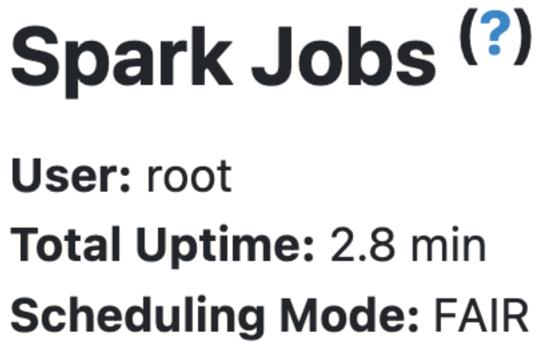


Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks
Active(3)	0	0.0 B / 4.1 GiB	0.0 B	2	0	0	53	53
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0
Total(3)	0	0.0 B / 4.1 GiB	0.0 B	2	0	0	53	53



Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores
driver	cluster-e4e7-m.us-central1-c.c.gentle-vista-437017-j7.internal:39301	Active	0	0.0 B / 1 GiB	0.0 B	0
1	cluster-e4e7-w-0.us-central1-c.c.gentle-vista-437017-j7.internal:39273	Active	0	0.0 B / 1.5 GiB	0.0 B	1
2	cluster-e4e7-w-1.us-central1-c.c.gentle-vista-437017-j7.internal:37297	Active	0	0.0 B / 1.5 GiB	0.0 B	1



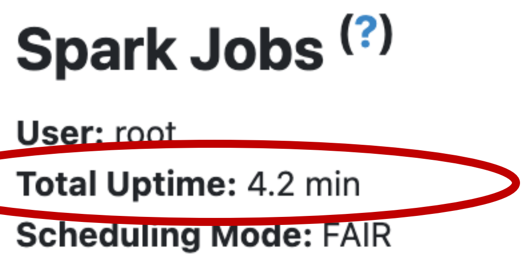
Spark Jobs (?)

User: root

Total Uptime: 2.8 min

Scheduling Mode: FAIR

Execution Breakdown in Local Mode



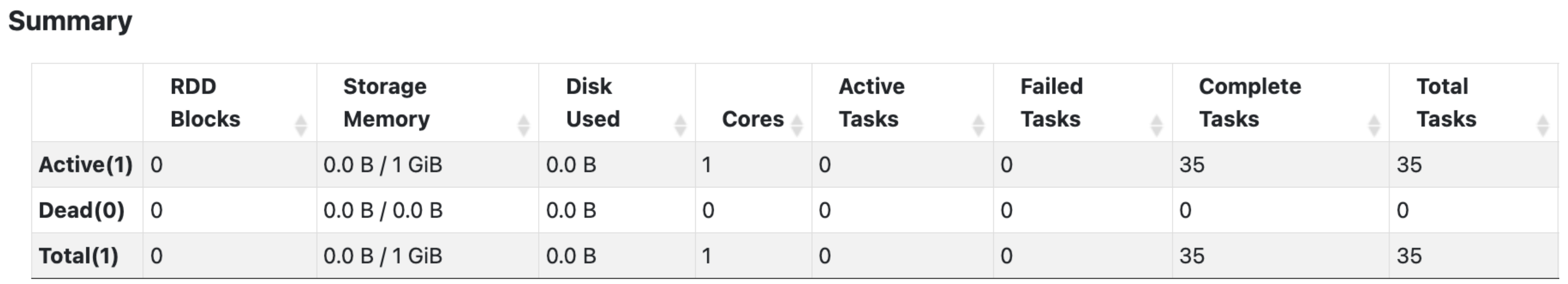
Spark Jobs (?)

User: root

Total Uptime: 4.2 min

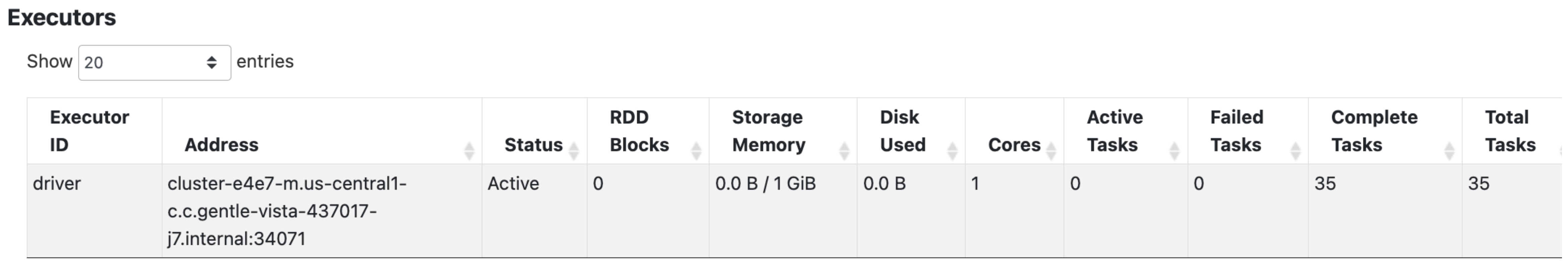
Scheduling Mode: FAIR

Slower than cluster mode!



Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks
Active(1)	0	0.0 B / 1 GiB	0.0 B	1	0	0	35	35
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0
Total(1)	0	0.0 B / 1 GiB	0.0 B	1	0	0	35	35



Executors

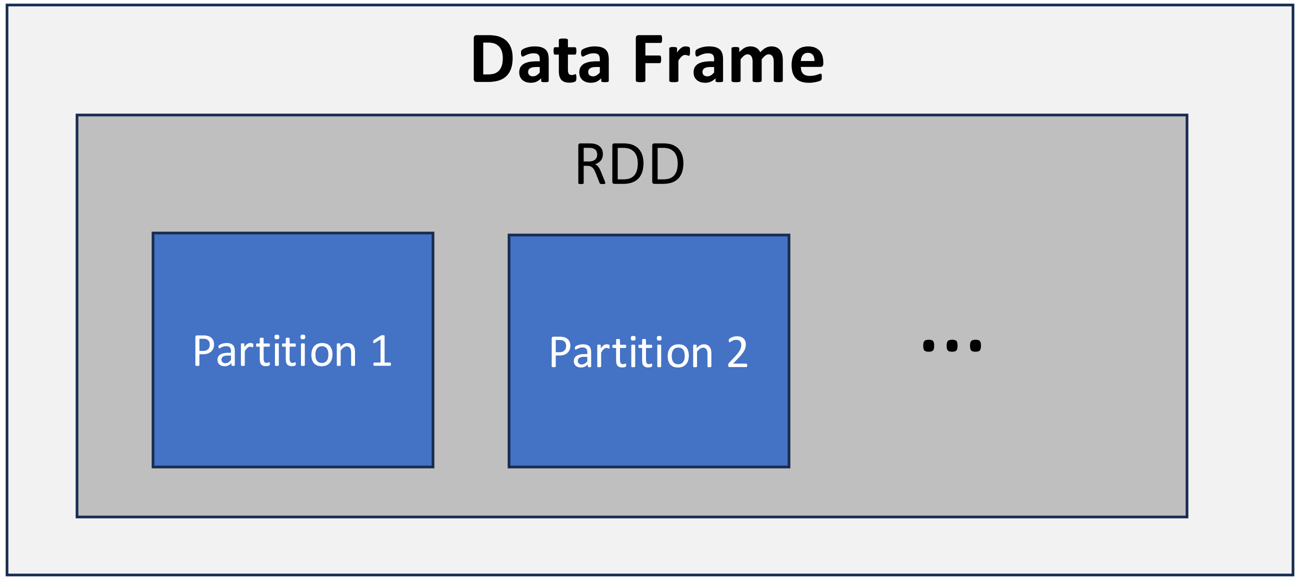
Show 20 entries

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks
driver	cluster-e4e7-m.us-central1-c.c.gentle-vista-437017-j7.internal:34071	Active	0	0.0 B / 1 GiB	0.0 B	1	0	0	35	35

ONLY one executer (the master node itself), no worker node used.

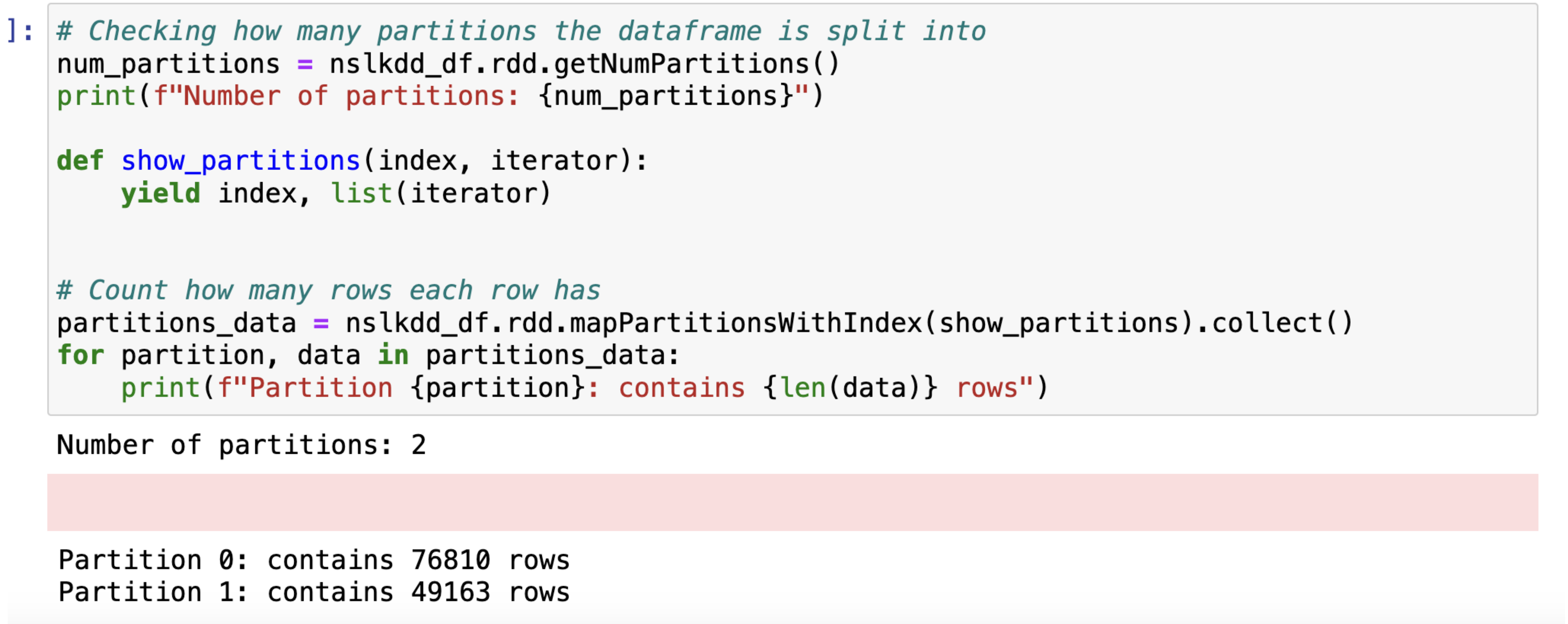
How distributed computation works in Spark?

Each data frame is built on top of an RDD (Resilient Distributed Datasets), which is split into partitions across nodes!



How distributed computation works in Spark?

Each data frame is built on top of an RDD (Resilient Distributed Datasets), which is split into partitions across nodes!



```
]: # Checking how many partitions the dataframe is split into
   num_partitions = nslkdd_df.rdd.getNumPartitions()
   print(f"Number of partitions: {num_partitions}")

   def show_partitions(index, iterator):
       yield index, list(iterator)


   # Count how many rows each row has
   partitions_data = nslkdd_df.rdd.mapPartitionsWithIndex(show_partitions).collect()
   for partition, data in partitions_data:
       print(f"Partition {partition}: contains {len(data)} rows")
```

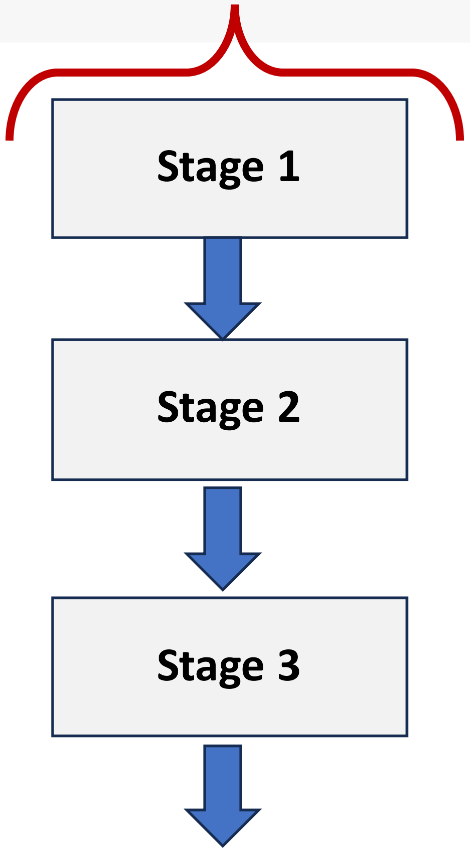
```
Number of partitions: 2
```

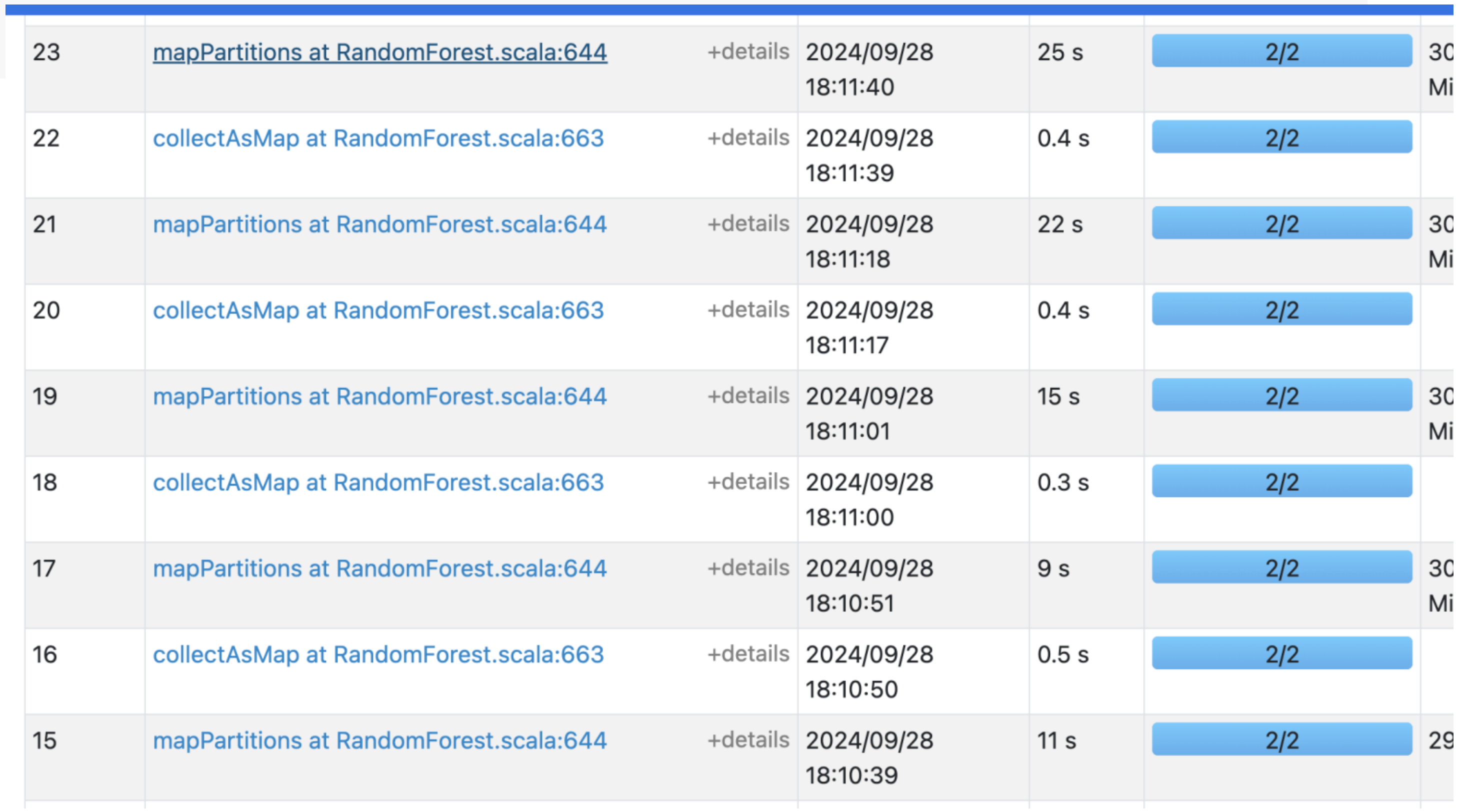
```
Partition 0: contains 76810 rows
Partition 1: contains 49163 rows
```

How distributed computation works in Spark?

For each line of code involving dataframe operation, Spark splits it into stages

```
rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'outcome',numTrees=500)
rf_model = rf.fit(nslkdd_df)
```





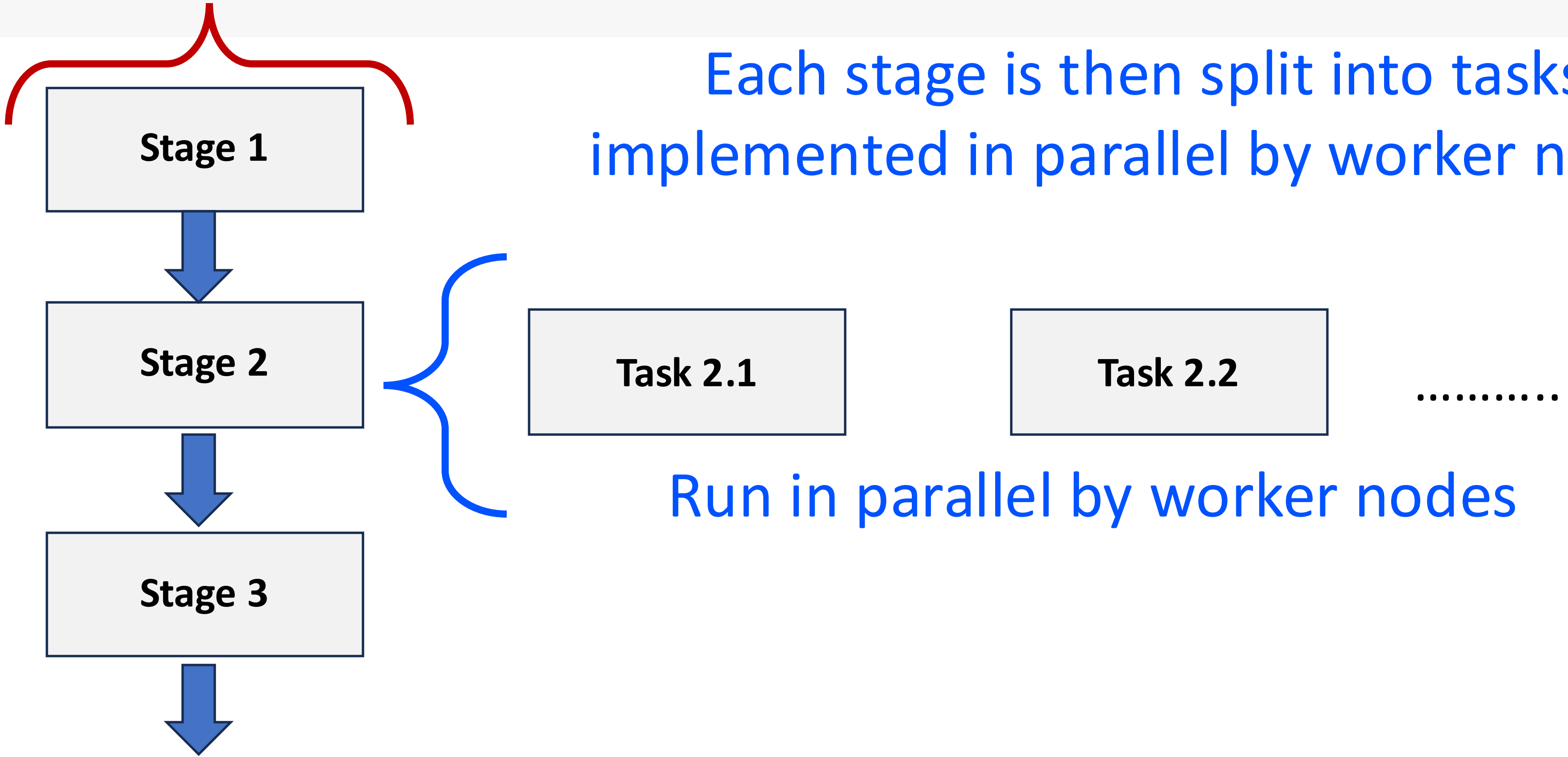
23	mapPartitions at RandomForest.scala:644	+details	2024/09/28 18:11:40	25 s	2/2	30 Mi
22	collectAsMap at RandomForest.scala:663	+details	2024/09/28 18:11:39	0.4 s	2/2	
21	mapPartitions at RandomForest.scala:644	+details	2024/09/28 18:11:18	22 s	2/2	30 Mi
20	collectAsMap at RandomForest.scala:663	+details	2024/09/28 18:11:17	0.4 s	2/2	
19	mapPartitions at RandomForest.scala:644	+details	2024/09/28 18:11:01	15 s	2/2	30 Mi
18	collectAsMap at RandomForest.scala:663	+details	2024/09/28 18:11:00	0.3 s	2/2	
17	mapPartitions at RandomForest.scala:644	+details	2024/09/28 18:10:51	9 s	2/2	30 Mi
16	collectAsMap at RandomForest.scala:663	+details	2024/09/28 18:10:50	0.5 s	2/2	
15	mapPartitions at RandomForest.scala:644	+details	2024/09/28 18:10:39	11 s	2/2	29

Actual stages based on Spark History

How distributed computation works in Spark?

For each line of code involving dataframe operation, Spark splits it into stages

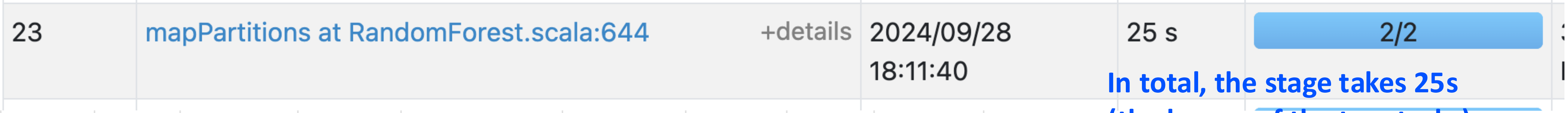
```
rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'outcome',numTrees=500)
rf_model = rf.fit(nslkdd_df)
```



Each stage is then split into tasks, implemented in parallel by worker nodes!
Stage 1
Stage 2
Stage 3
Task 2.1
Task 2.2
....
Run in parallel by worker nodes

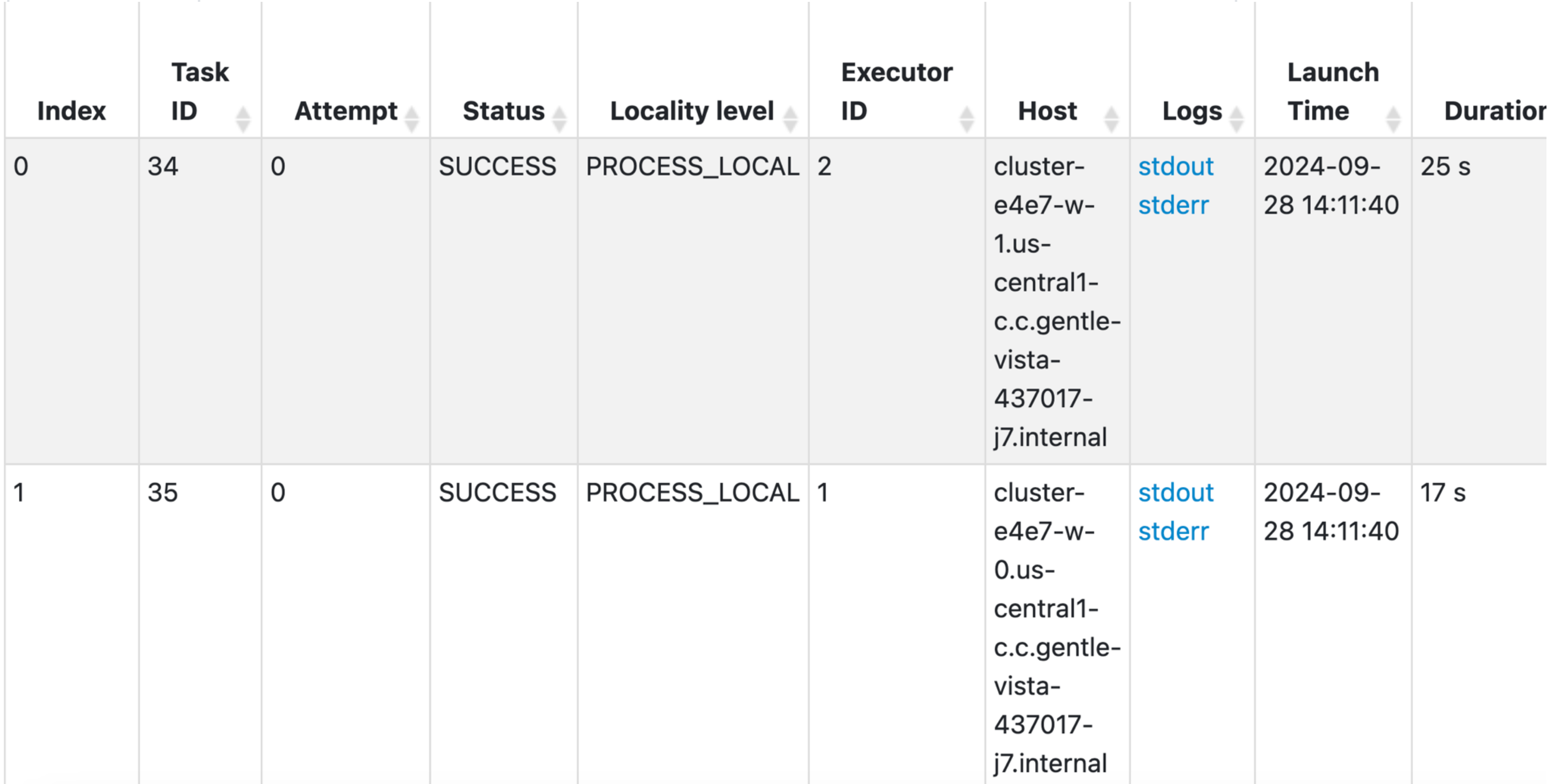
How distributed computation works in Spark?

Example: Task decomposition of stage 23



23	mapPartitions at RandomForest.scala:644	+details	2024/09/28 18:11:40	25 s	2/2

In total, the stage takes 25s (the longer of the two tasks)



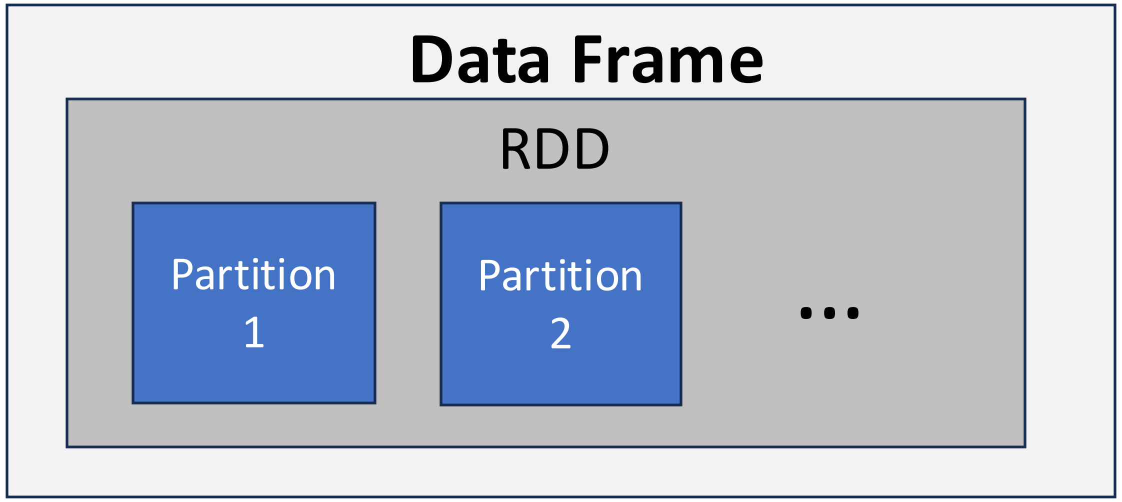
Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duratior
0	34	0	SUCCESS	PROCESS_LOCAL	2	cluster-e4e7-w-1.us-central1-c.c.gentle-vista-437017-j7.internal	stdout stderr	2024-09-28 14:11:40	25 s
1	35	0	SUCCESS	PROCESS_LOCAL	1	cluster-e4e7-w-0.us-central1-c.c.gentle-vista-437017-j7.internal	stdout stderr	2024-09-28 14:11:40	17 s

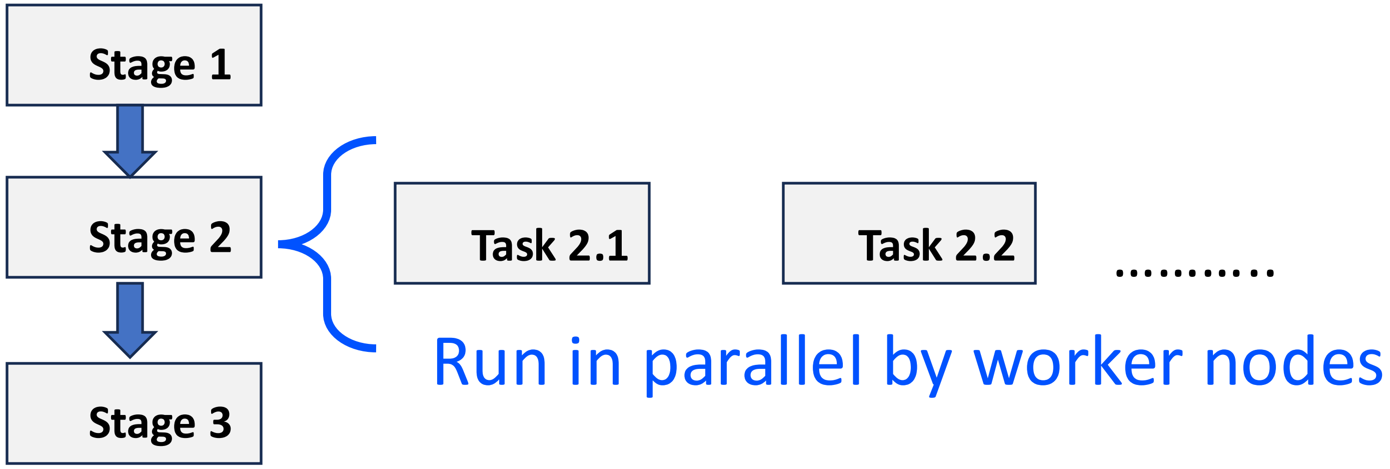
Task 0 takes 25s

Task 1 takes 17s

Summary

- Each dataframe is an RDD, which is split into partitions and stored in different nodes
- Each dataframe operation is broken down into stages. Each stage is split into tasks, and the tasks are run in parallel by different nodes.

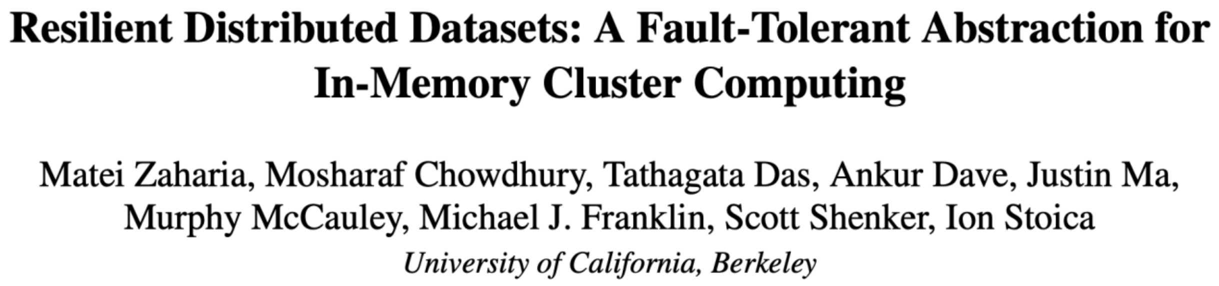




Further Details

How does Spark split a stage into tasks that can run in parallel?

- All dataframe manipulation boils down to a set of basic RDD transformations and actions, all of which can be parallelized.
- Three most important RDD transformations and actions
  - MAP
  - REDUCE
  - FILTER
- (OPTIONAL) See this lecture's code for examples of MAP/REDUCE/FILTER
- (OPTIONAL) Read the Spark Paper



Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley