

# Machine Learning for the Spark Developer

Paige Liu, Microsoft


**#Dev2SAIS**

# Agenda

- A simple ML example in R
- Spark MLlib overview
- Implement the same example in Spark MLlib

# An example in R

- Engine Remaining Useful Life (RUL) prediction


 Azure AI Gallery

[Browse all](#) [Industries](#) [Solutions](#) [Projects](#) [Models](#) [Experiments](#) [Custom Modules](#) [More](#)

COLLECTION

## Predictive Maintenance Template

By AzureML Team for Microsoft • September 28, 2015

 21 likes

[Twitter](#) [LinkedIn](#) [Email](#)


**Summary**

This template demonstrate how to build and deploy predictive maintenance models to predict asset failures.


**Description**

Predictive maintenance encompasses a variety of topics, including but not limited to: failure prediction, failure diagnosis (root cause analysis), failure detection, failure type classification, and recommendation of mitigation or maintenance actions after failure. As part of the Azure Machine Learning offering, Microsoft provides a template that helps data scientists easily build and deploy a predictive maintenance solution. **This predictive maintenance template focuses on the techniques used to predict when an in-service machine will fail, so that maintenance can be planned in advance.** The template includes a collection of pre-configured machine learning modules, as well as custom R scripts in the *Execute R Script* module, to enable an end-to-end solution from data processing to deploying of the machine learning model.


[Read full description](#)



+ Add to Collection

 23209 views

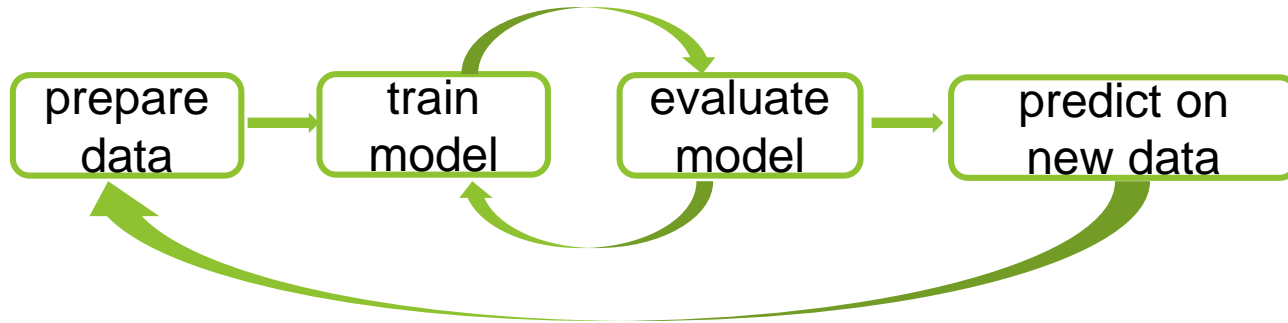
**RELATED ITEMS**

 **Predictive Maintenance Template**  
EXPERIMENT by Yan Zhang

Data: [NASA Ames Prognostics Data Repository](#)

# R sample

# Machine learning workflow



# Algorithms in MLlib

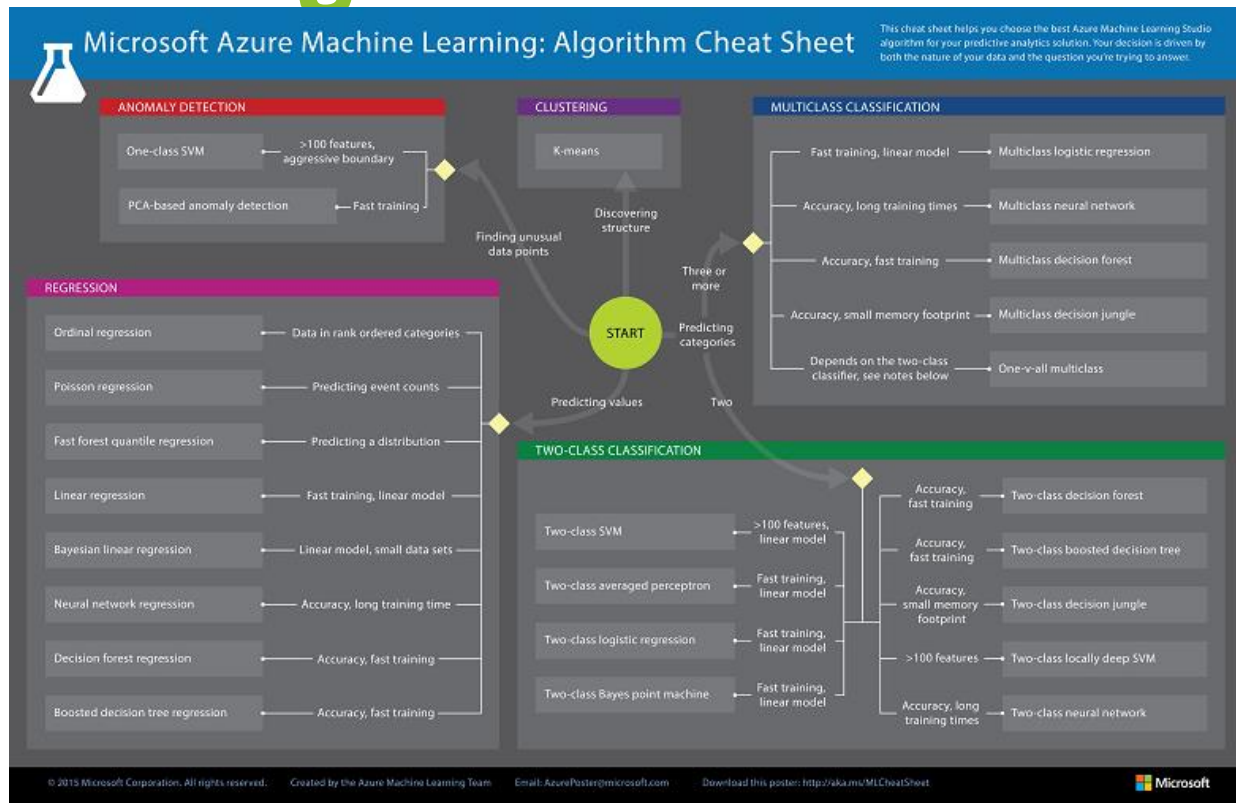
- Classification – predict categories
- Regression – predict numeric values

} Supervised  
learning

- Clustering – group similar items
- Collaborative filtering – recommendation engine
- Frequent pattern mining - anomaly detection

} Unsupervised  
learning

# Which algorithm to use?



<https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-cheat-sheet>

# MLlib Concepts

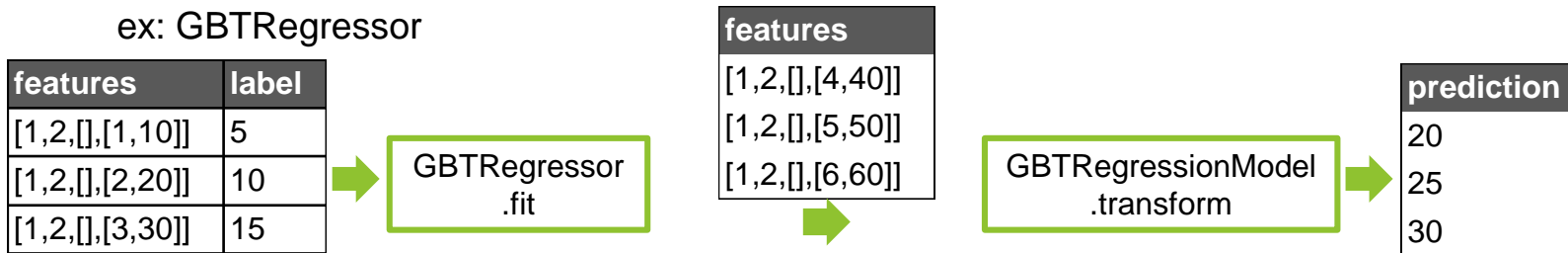
- **Transformers:** data frame -> .transform -> data frame

ex: VectorAssembler



- **Estimators:** data frame -> .fit -> .transform -> data frame

ex: GBTRegressor



- **Pipeline:** ML workflow consists of a series of transformers and estimators



# Prepare training data

Engine id	Cycle	Setting1	Setting2	Setting3	Sensor1	...	Sensor21
1	1	-0.0007	-0.0004	100.0	518.67	...	23.4190
1	2	0.0019	-0.0003	100.0	518.67	...	23.4236
...							



Engine id	Cycle	Sensor9	Sensor11	Sensor14	Sensor15		RUL=max – cur cycle
1	1	9046.19	47.47	8138.62	8.4195	+	191
1	2	9044.07	47.49	8131.49	8.4318		190
...							...

# Train the model



features	RUL
[1.0, 9046.19, 47.47, 8138.62, 8.4195]	191
[2.0, 9044.07, 47.49, 8131.49, 8.4318]	190
...	...



```
"GBTRegressionModel (uid=gbtr_2d5316e37e51) with 50 trees
Tree 0 (weight 1.0):
  If (feature 0 <= 96.5)
    If (feature 0 <= 51.5)
      If (feature 0 <= 25.5)
        If (feature 2 <= 47.35499954223633)
          If (feature 4 <= 8.418050289154053)
            Predict: 205.94230769230768
          Else (feature 4 > 8.418050289154053)
            Predict: 191.46132596685084
        Else (feature 2 > 47.35499954223633)
          If (feature 0 <= 13.5)
            Predict: 190.72403100775193
          Else (feature 0 > 13.5)
            Predict: 178.81116584564862
```

```
val vectorAssembler = new VectorAssembler()
  .setInputCols(Array(
    "cycle", "s9", "s11", "s14", "s15"))
  .setOutputCol("features")
```

```
val gbtRegressor = new GBTRegressor()
  .setLabelCol("RUL")
  .setFeaturesCol("features")
```

```
val pipeline = new Pipeline()
  .setStages(Array(
    vectorAssembler, gbtRegressor))
val gbtRegressionModel = pipeline
  .fit(train_df)
```

# Prepare test data

Engine id	Cycle	Setting1	Setting2	Setting3	Sensor1	...	Sensor21		RUL
1	1	0.0023	0.0003	100.0	518.67	...	23.3735		112
1	2	-0.0027	-0.0003	100.0	518.67	...	23.3916	+	98
...	...	...	...	...	...	...	...		...



Engine id	Last cycle	Sensor9	Sensor11	Sensor14	Sensor15	RUL
1	31	9056.4	47.23	8130.11	8.4024	112
2	49	9044.07	47.49	8131.49	8.4318	98
...	...	...	...	...	...	...

# Evaluate the model



features	RUL
[31, 9056.4, 47.23, 8130.11, 8.4024]	112
[49, 9044.07, 47.49, 8131.49, 8.4318]	98
...	...



prediction
196.76
152.64
...



**root mean squared error**

$\text{sqrt}(\text{mean}((\text{labeled}-\text{predicted})^2)) = 29.55$

```
val predictions = gbtRegressionModel  
  .transform(test_df)
```

```
val evaluator = new RegressionEvaluator()  
  .setLabelCol("RUL")  
  .setPredictionCol("prediction")  
  .setMetricName("rmse")  
val rootMeanSquaredError = evaluator  
  .evaluate(predictions)
```

# Tuning the model

- Hyper-parameter tuning

maxIter	maxDepth	stepSize
10	5	0.1
10	5	0.2
10	10	0.1
...	...	...

```
val paramGrid = new ParamGridBuilder()  
  .addGrid(gbtRegressor.maxIter, Array(10, 50, 100))  
  .addGrid(gbtRegressor.maxDepth, Array(5, 10))  
  .addGrid(gbtRegressor.stepSize, Array(0.1, 0.2))  
  .build
```

- Cross validation

maxIter	maxDepth	stepSize	train_df		
10	5	0.1	train	train	test
10	5	0.1	train	test	train
10	5	0.1	test	train	train
10	5	0.2	train	train	test
...	...	...	...	...	...

```
val crossValidator = new CrossValidator()  
  .setEstimator(pipeline)  
  .setEvaluator(evaluator)  
  .setEstimatorParamMaps(paramGrid)  
  .setNumFolds(3)  
  .setParallelism(3)  
val crossValidatorModel = crossValidator  
  .fit(train_df)
```

# Predict on new data in application

- Save the model

```
crossValidatorModel.bestModel  
  .write  
  .overwrite()  
  .save("/path/to/model")
```

- Load the model to make predictions

```
val new_df = ...  
val model = PipelineModel  
  .load ("/path/to/model")  
val predictions = model  
  .transform(new_df)
```

- Load the model to non-Spark applications
  - MLeap: serialization format + execution engine
  - Databricks ML Model Export

# Resources

- GitHub repo containing the sample:  
<https://github.com/liupeirong/sparksummit2018ml>
  - Customer churn: binary classification
  - Iris: multi-class classification
  - Predictive maintenance: regression
- Documentation on the sample: <https://gallery.azure.ai/Collection/Predictive-Maintenance-Template-3>
- NASA data repository hosting the sample data:  
<https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/>

# Thank You