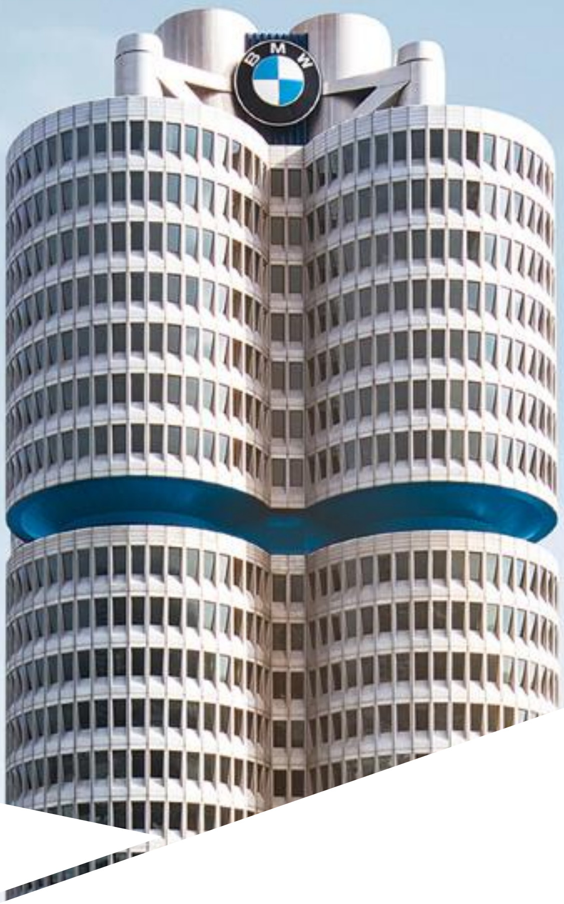
BUILDING CUSTOM ML PIPELINESTAGES FOR FEATURE SELECTION.

SPARK SUMMIT EUROPE 2017.



Kaminski, Schlegel | Oct. 25, 2017

BMW GROUP

THE NEXT 100 YEARS

MINI

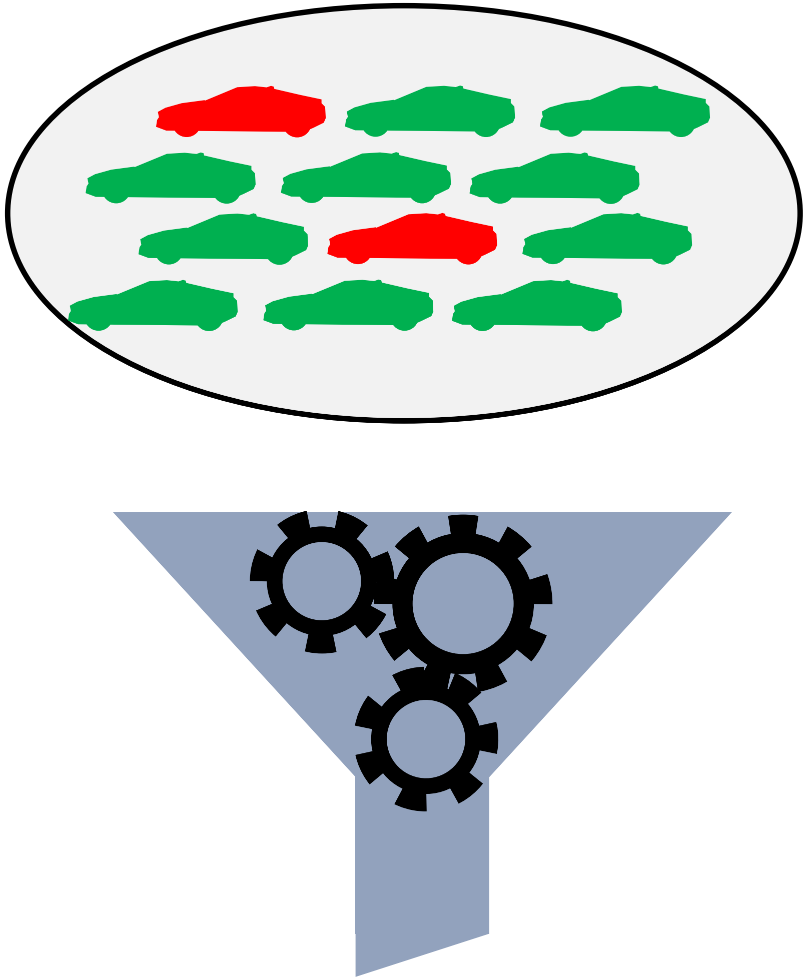
Rolls-Royce Motor Cars Limited

WHAT YOU WILL LEARN DURING THIS SESSION.

- How data-driven car diagnostics look like at BMW.
- Get a good understanding of the most important elements in Spark ML PipelineStages (on a feature selection example).
  - Attention: There will be Scala code examples!
- How to use spark-FeatureSelection in your Spark ML Pipeline.
- The impact of feature selection on learning performance and the understanding of the big data black box.

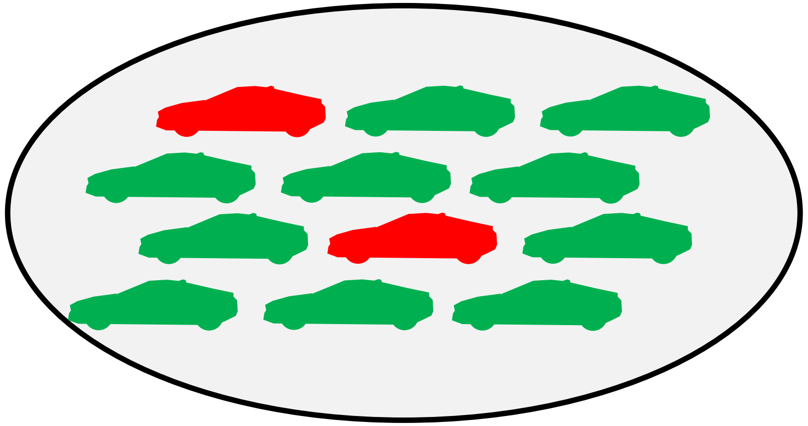
MOTIVATION.

– #3 contributor to warranty incidents for OEMs are “no trouble found” cases. [1]

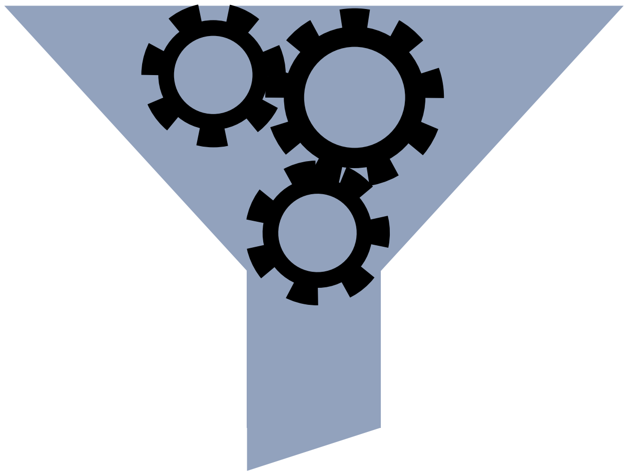


[1] BearingPoint, Global Automotive Warranty Survey Report 2009

MOTIVATION.

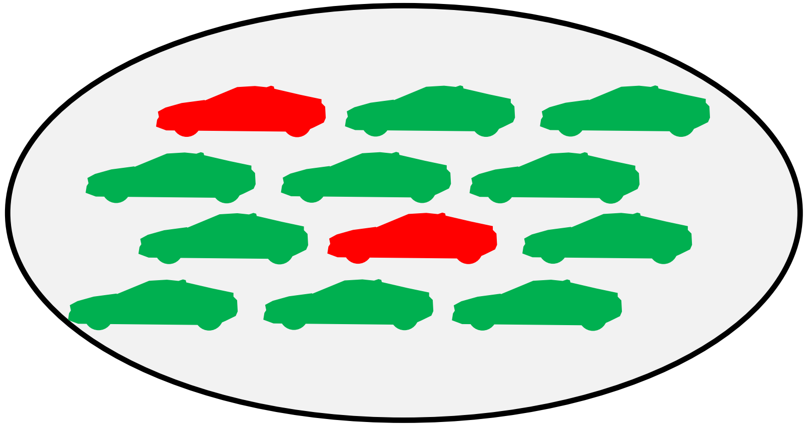


- #3 contributor to warranty incidents for OEMs are “no trouble found” cases. [1]
- Potential root causes:
  - Manually formalized expert knowledge cannot cope with the vast number of possibilities.
  - Cars are getting more and more complex (hybridization, connectivity).
  - Less experienced workshop staff in evolving markets.

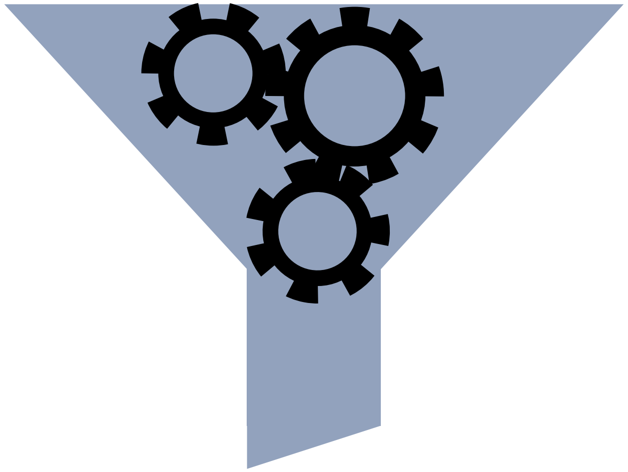


[1] BearingPoint, Global Automotive Warranty Survey Report 2009

MOTIVATION.



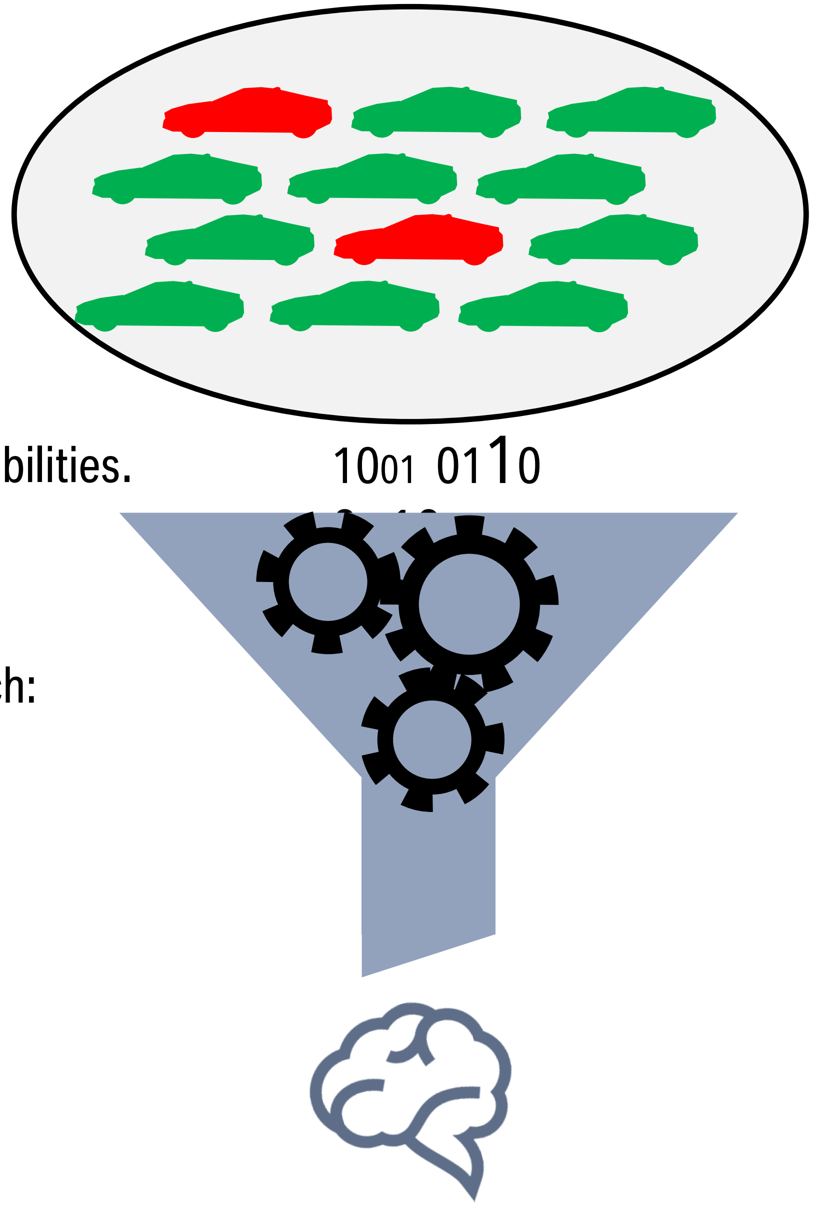
- #3 contributor to warranty incidents for OEMs are “no trouble found” cases. [1]
- Potential root causes:
  - Manually formalized expert knowledge cannot cope with the vast number of possibilities.
  - Cars are getting more and more complex (hybridization, connectivity).
  - Less experienced workshop staff in evolving markets.
- Improve three workflows at once by shifting from a manual to a **data driven** approach:



[1] BearingPoint, Global Automotive Warranty Survey Report 2009

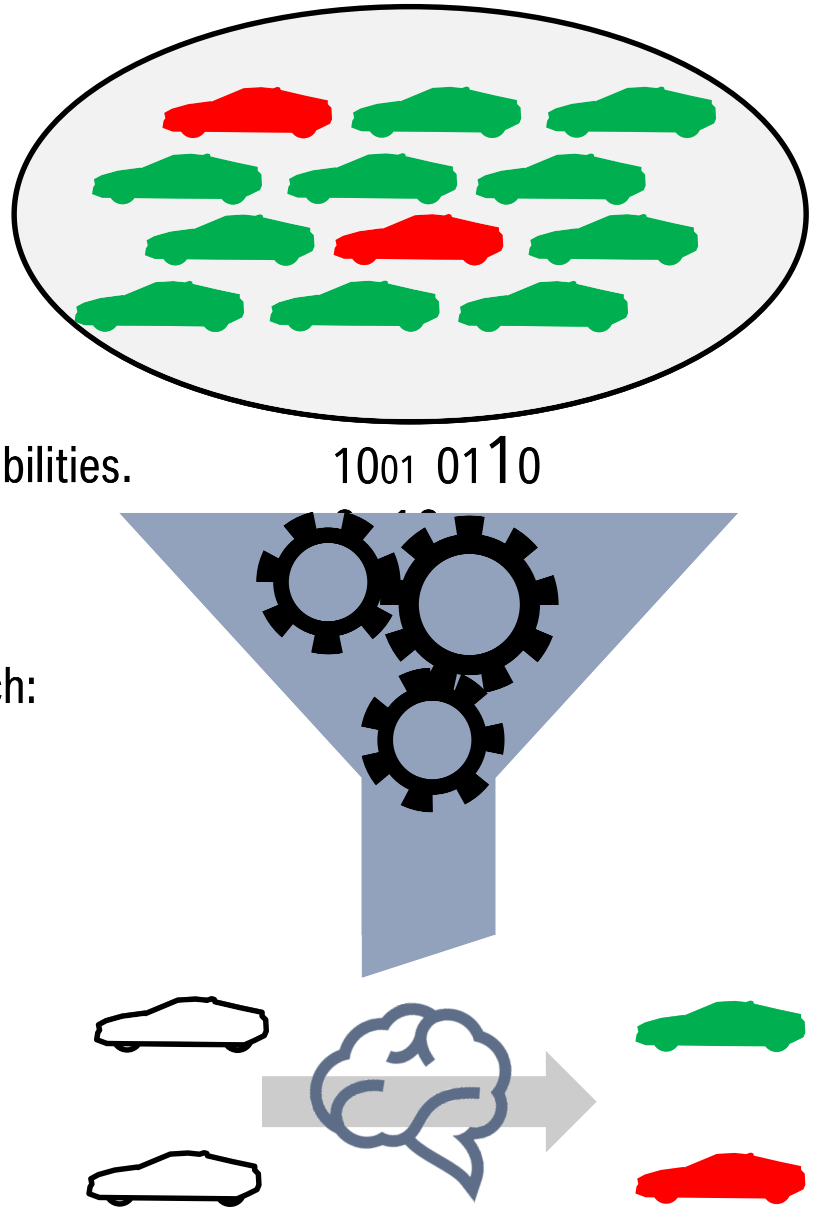
MOTIVATION.

- #3 contributor to warranty incidents for OEMs are “no trouble found” cases. [1]
- Potential root causes:
  - Manually formalized expert knowledge cannot cope with the vast number of possibilities.
  - Cars are getting more and more complex (hybridization, connectivity).
  - Less experienced workshop staff in evolving markets.
- Improve three workflows at once by shifting from a manual to a **data driven** approach:
  - Automatic knowledge generation.



[1] BearingPoint, Global Automotive Warranty Survey Report 2009

MOTIVATION.



- #3 contributor to warranty incidents for OEMs are “no trouble found” cases. [1]
- Potential root causes:
  - Manually formalized expert knowledge cannot cope with the vast number of possibilities.
  - Cars are getting more and more complex (hybridization, connectivity).
  - Less experienced workshop staff in evolving markets.
- Improve three workflows at once by shifting from a manual to a **data driven** approach:
  - Automatic knowledge generation.
  - Automatic workshop diagnostics.
  - Predictive maintenance.

[1] BearingPoint, Global Automotive Warranty Survey Report 2009

THE DATASET AND ITS CHALLENGES.

MV_S	MV_0	…	MV_4000	MV_BG	EE_TP	SC_IP	SC_1	SC_2	DTC_PU	DTC_1	DTC_2	CP	Label
44	3	…	20	-0.06	false	2	77	27	false		true	v.10	false
72	36	…	73	-0.01	false		16	29			false	v.10	false
100	4	…	16	-0.02	true		45	1		false	false	v.10	false
44	14	…	54	-0.02	true		76				false	v.10	true
95	34	…	73	-0.07	false		80	22		false	false	v.10	false
16	50	…	33	-0.02	true		61	93	false	false	false	v.11	false
	4	…	27	-0.09	false		59	91			false	v.10	false
	48	…	20	-0.07	false		32	31			false	v.10	false
88	60	…	72	-0.01	true	1.9	96	53	true	false	true	v.10	false
27	14	…	88		false		73	14			false	v.10	false

THE DATASET AND ITS CHALLENGES.

High dimensional featurespace (7000 features +)

MV_S	MV_0	…	MV_4000	MV_BG	EE_TP	SC_IP	SC_1	SC_2	DTC_PU	DTC_1	DTC_2	CP	Label
44	3	…	20	-0.06	false	2	77	27	false		true	v.10	false
72	36	…	73	-0.01	false		16	29			false	v.10	false
100	4	…	16	-0.02	true		45	1		false	false	v.10	false
44	14	…	54	-0.02	true		76				false	v.10	true
95	34	…	73	-0.07	false		80	22		false	false	v.10	false
16	50	…	33	-0.02	true		61	93	false	false	false	v.11	false
	4	…	27	-0.09	false		59	91			false	v.10	false
	48	…	20	-0.07	false		32	31			false	v.10	false
88	60	…	72	-0.01	true	1.9	96	53	true	false	true	v.10	false
27	14	…	88		false		73	14			false	v.10	false

THE DATASET AND ITS CHALLENGES.

High dimensional featurespace (7000 features +)

MV_S	MV_0	…	MV_4000	MV_BG	EE_TP	SC_IP	SC_1	SC_2	DTC_PU	DTC_1	DTC_2	CP	Label
44	3	…	20	-0.06	false	2	77	27	false		true	v.10	false
72	36	…	73	-0.01	false		16	29			false	v.10	false
100	4	…	16	-0.02	true		45	1		false	false	v.10	false
44	14	…	54	-0.02	true		76				false	v.10	true
95	34	…	73	-0.07	false		80	22		false	false	v.10	false
16	50	…	33	-0.02	true		61	93	false	false	false	v.11	false
	4	…	27	-0.09	false		59	91			false	v.10	false
	48	…	20	-0.07	false		32	31			false	v.10	false
88	60	…	72	-0.01	true	1.9	96	53	true	false	true	v.10	false
27	14	…	88		false		73	14			false	v.10	false

High sparsity

THE DATASET AND ITS CHALLENGES.

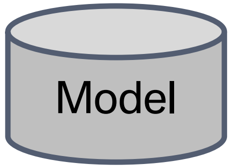
High dimensional featurespace (7000 features +)

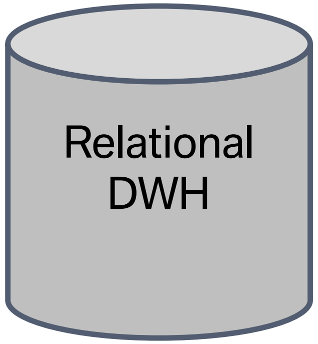
High class imbalance

MV_S	MV_0	…	MV_4000	MV_BG	EE_TP	SC_IP	SC_1	SC_2	DTC_PU	DTC_1	DTC_2	CP	Label
44	3	…	20	-0.06	false	2	77	27	false		true	v.10	false
72	36	…	73	-0.01	false		16	29			false	v.10	false
100	4	…	16	-0.02	true		45	1		false	false	v.10	false
44	14	…	54	-0.02	true		76				false	v.10	true
95	34	…	73	-0.07	false		80	22		false	false	v.10	false
16	50	…	33	-0.02	true		61	93	false	false	false	v.11	false
	4	…	27	-0.09	false		59	91			false	v.10	false
	48	…	20	-0.07	false		32	31			false	v.10	false
88	60	…	72	-0.01	true	1.9	96	53	true	false	true	v.10	false
27	14	…	88		false		73	14			false	v.10	false

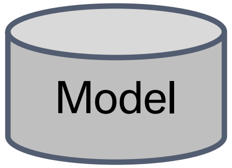
High sparsity

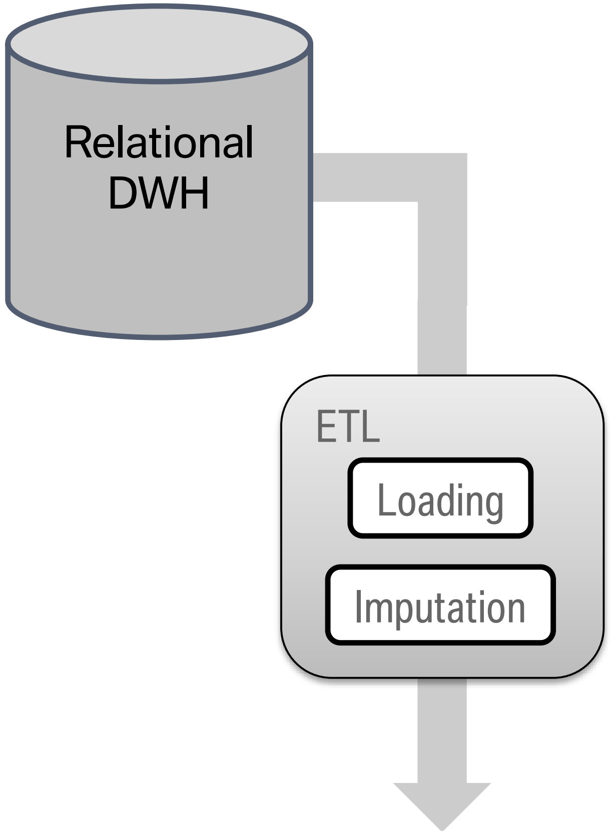
SPARK PIPELINE.





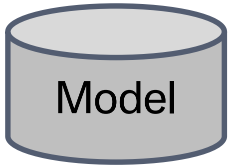
SPARK PIPELINE.

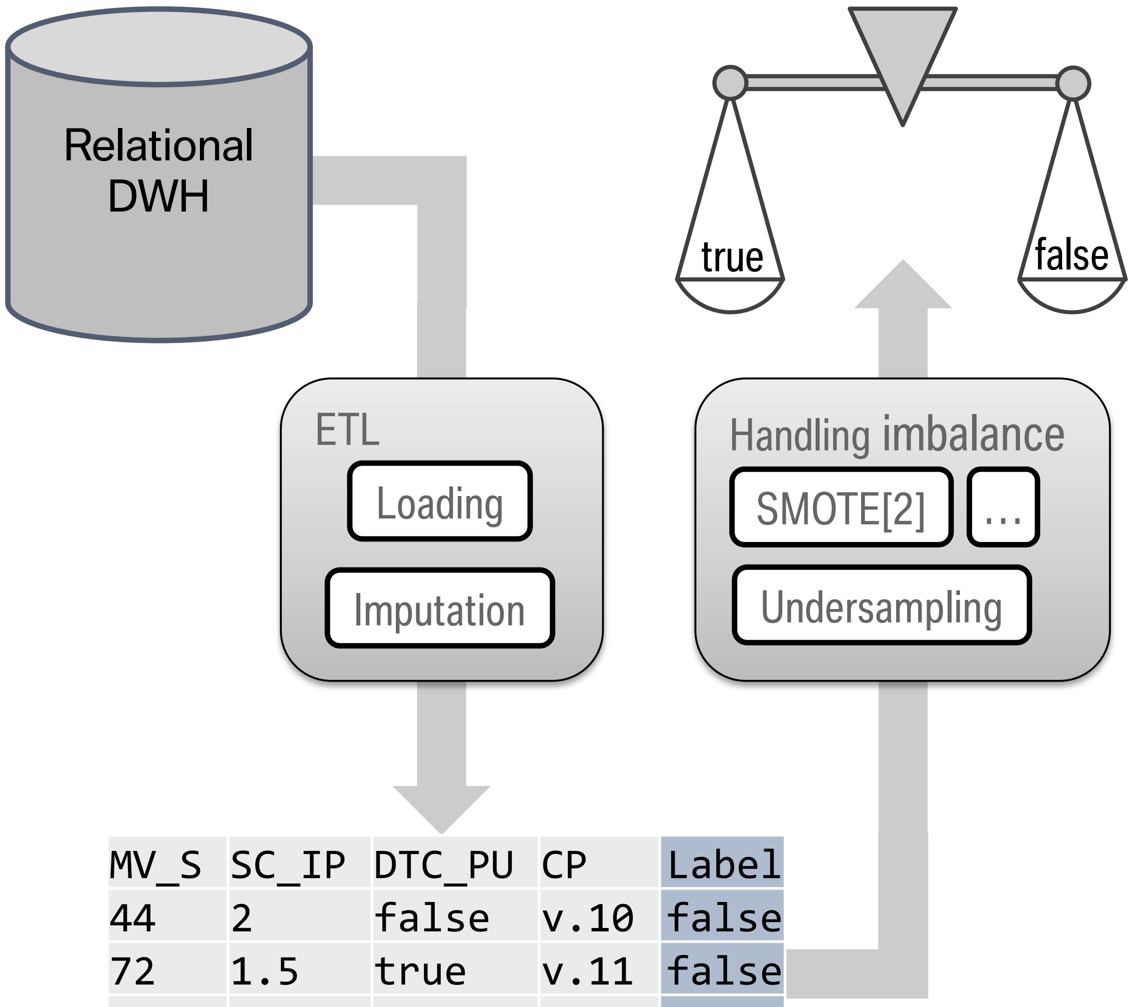




MV_S	SC_IP	DTC_PU	CP	Label
44	2	false	v.10	false
72	1.5	true	v.11	false
23	1.4	false	v.11	false
44	1.5	true	v.10	true

SPARK PIPELINE.

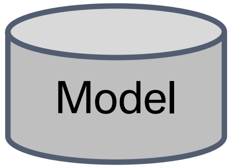


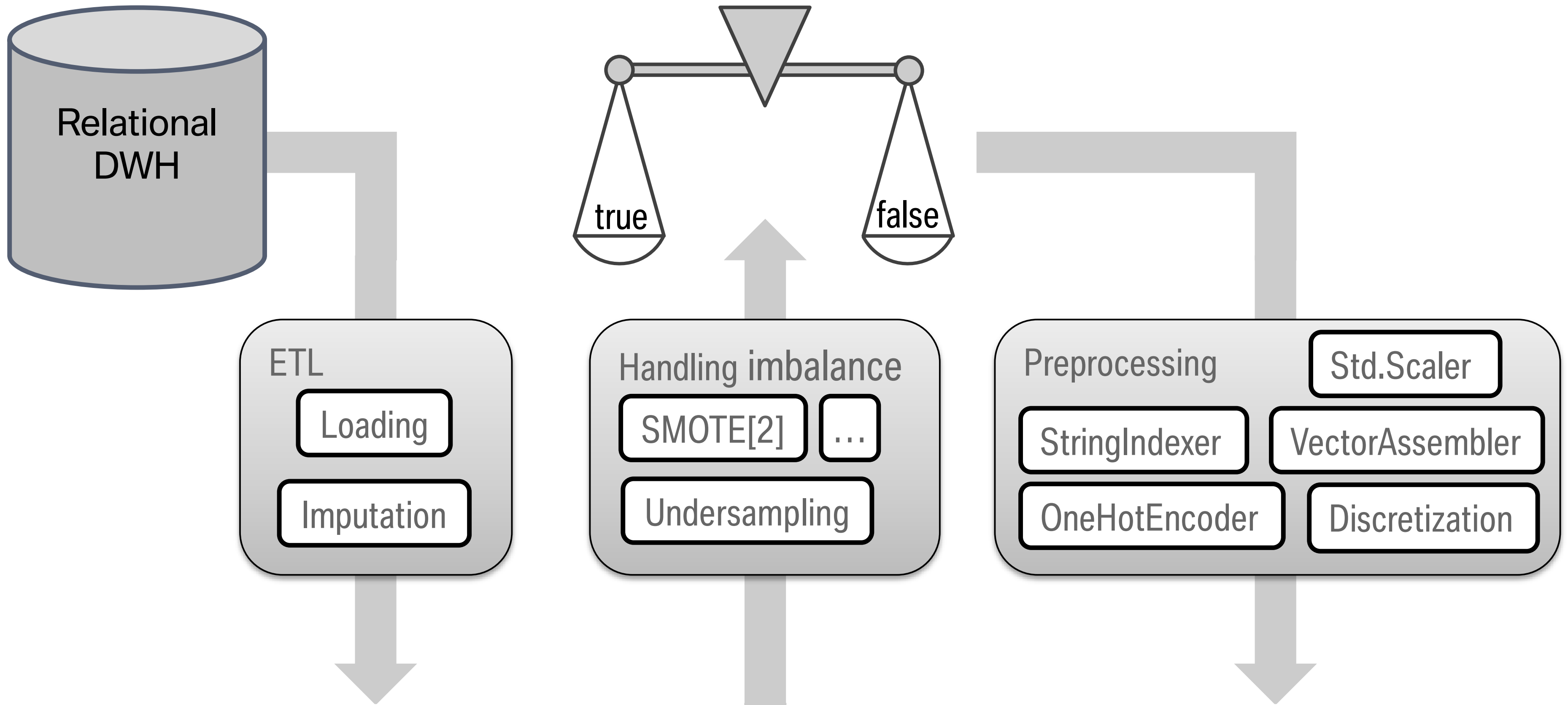


MV_S	SC_IP	DTC_PU	CP	Label
44	2	false	v.10	false
72	1.5	true	v.11	false
23	1.4	false	v.11	false
44	1.5	true	v.10	true

[2] Chawla et al.: SMOTE: Synthetic Minority Over-sampling Technique

SPARK PIPELINE.



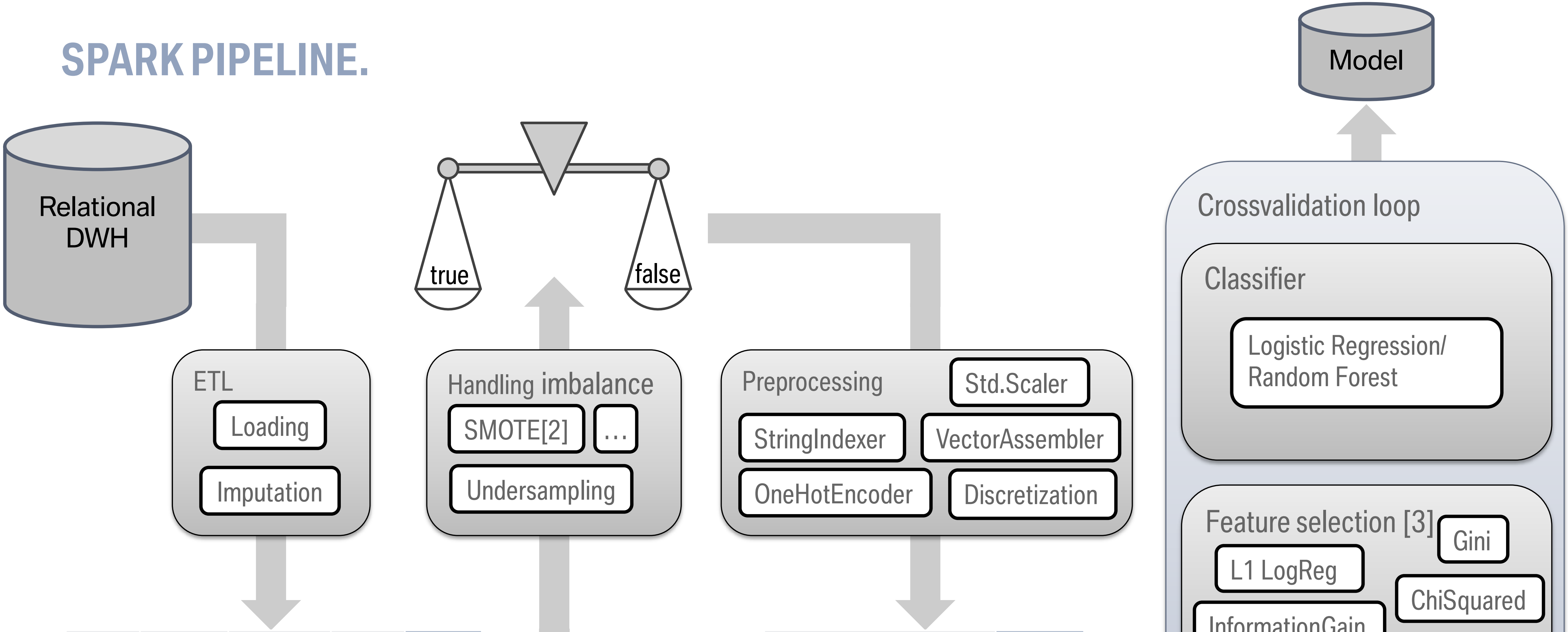


MV_S	SC_IP	DTC_PU	CP	Label
44	2	false	v.10	false
72	1.5	true	v.11	false
23	1.4	false	v.11	false
44	1.5	true	v.10	true

Features	Label
[0.34,0.8,0,1]	0.0
[0.7,0.4,1,0]	0.0
[0.31,0.35,1,0]	1.0
[0.3,0.4,1.1]	1.0

[2] Chawla et al.: SMOTE: Synthetic Minority Over-sampling Technique

SPARK PIPELINE.

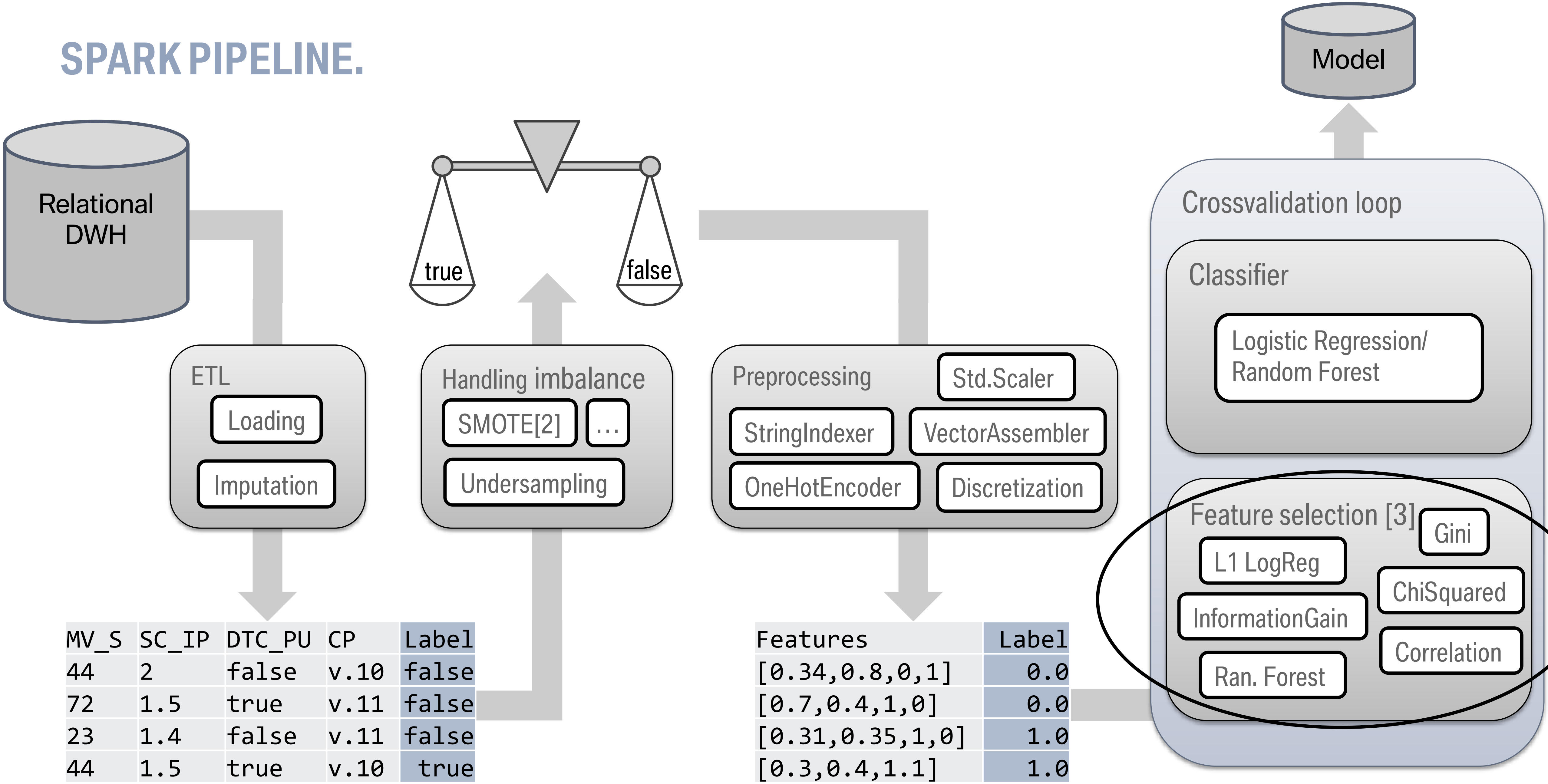


MV_S	SC_IP	DTC_PU	CP	Label
44	2	false	v.10	false
72	1.5	true	v.11	false
23	1.4	false	v.11	false
44	1.5	true	v.10	true

Features	Label
[0.34,0.8,0,1]	0.0
[0.7,0.4,1,0]	0.0
[0.31,0.35,1,0]	1.0
[0.3,0.4,1.1]	1.0

[2] Chawla et al.: SMOTE: Synthetic Minority Over-sampling Technique
[3]: Schlegel et al.: Design and optimization of an autonomous feature selection pipeline for high dimensional, heterogeneous feature spaces.

SPARK PIPELINE.



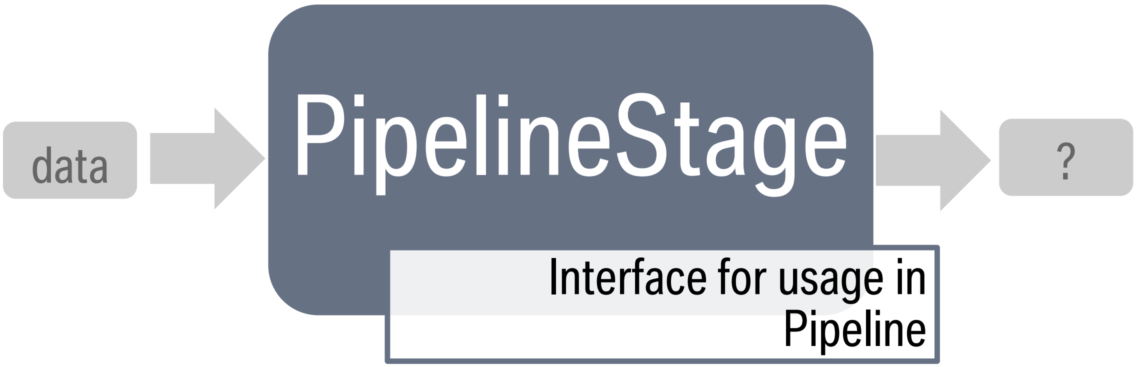
MV_S	SC_IP	DTC_PU	CP	Label
44	2	false	v.10	false
72	1.5	true	v.11	false
23	1.4	false	v.11	false
44	1.5	true	v.10	true

Features	Label
[0.34,0.8,0,1]	0.0
[0.7,0.4,1,0]	0.0
[0.31,0.35,1,0]	1.0
[0.3,0.4,1.1]	1.0

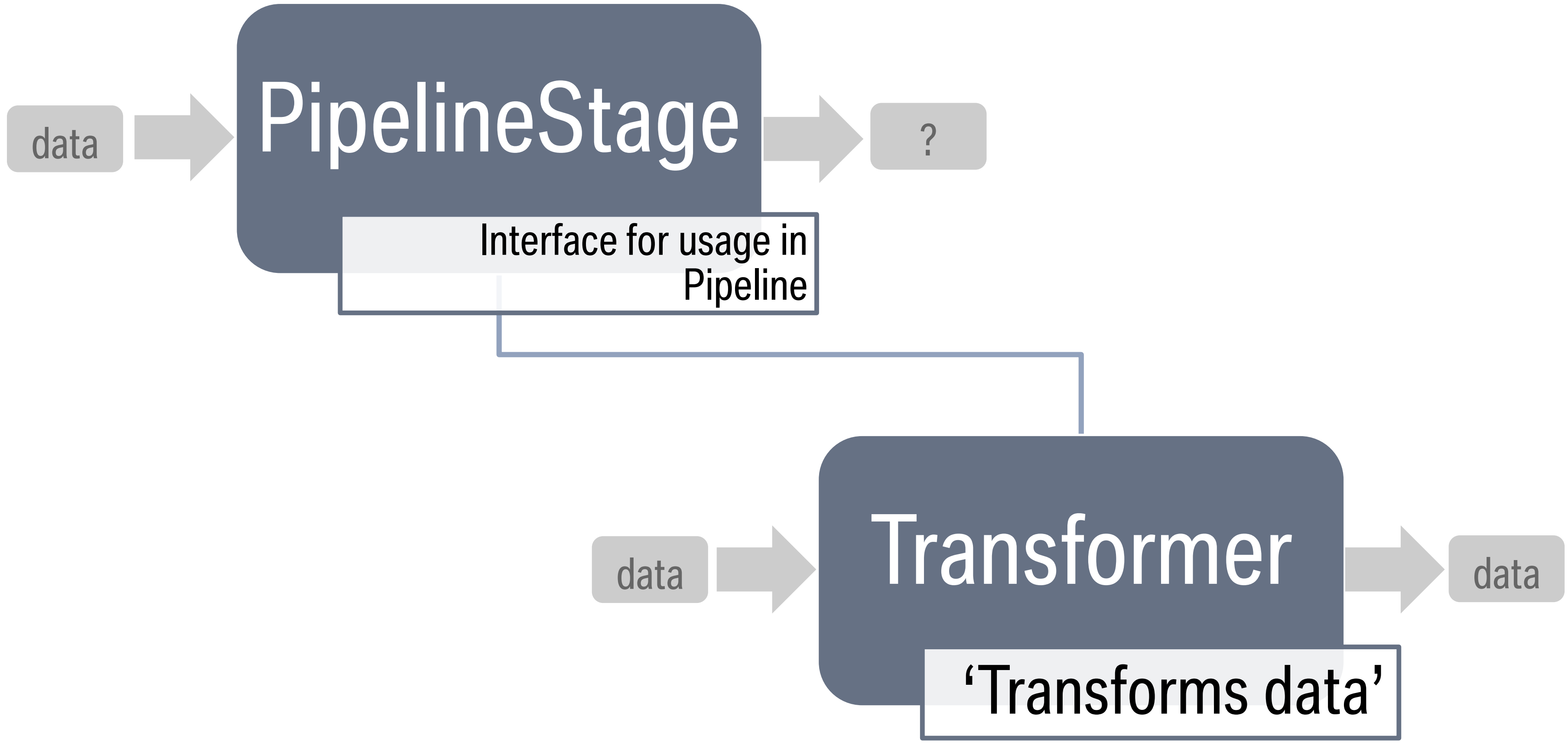
[2] Chawla et al.: SMOTE: Synthetic Minority Over-sampling Technique

[3]: Schlegel et al.: Design and optimization of an autonomous feature selection pipeline for high dimensional, heterogeneous feature spaces.

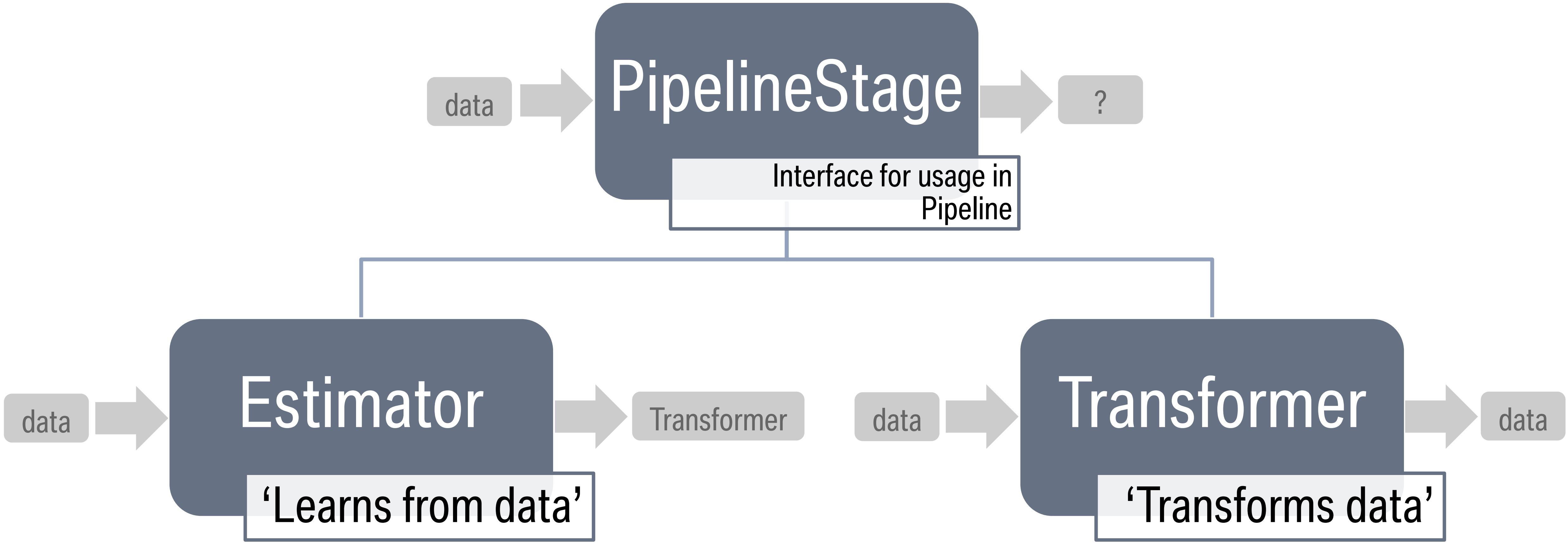
SPARK PIPELINE API.



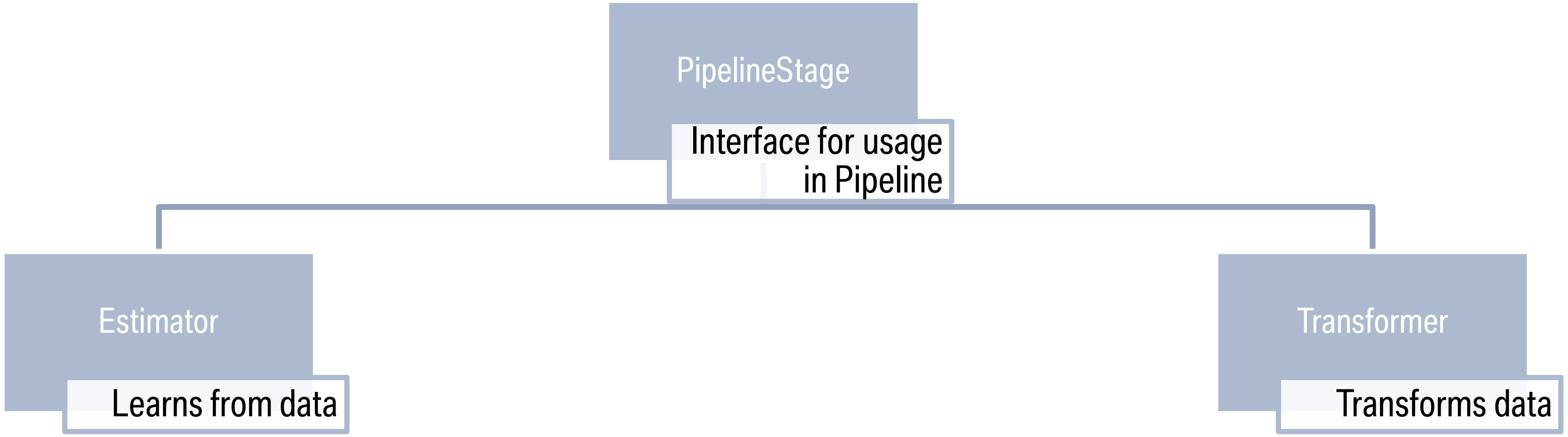
SPARK PIPELINE API.



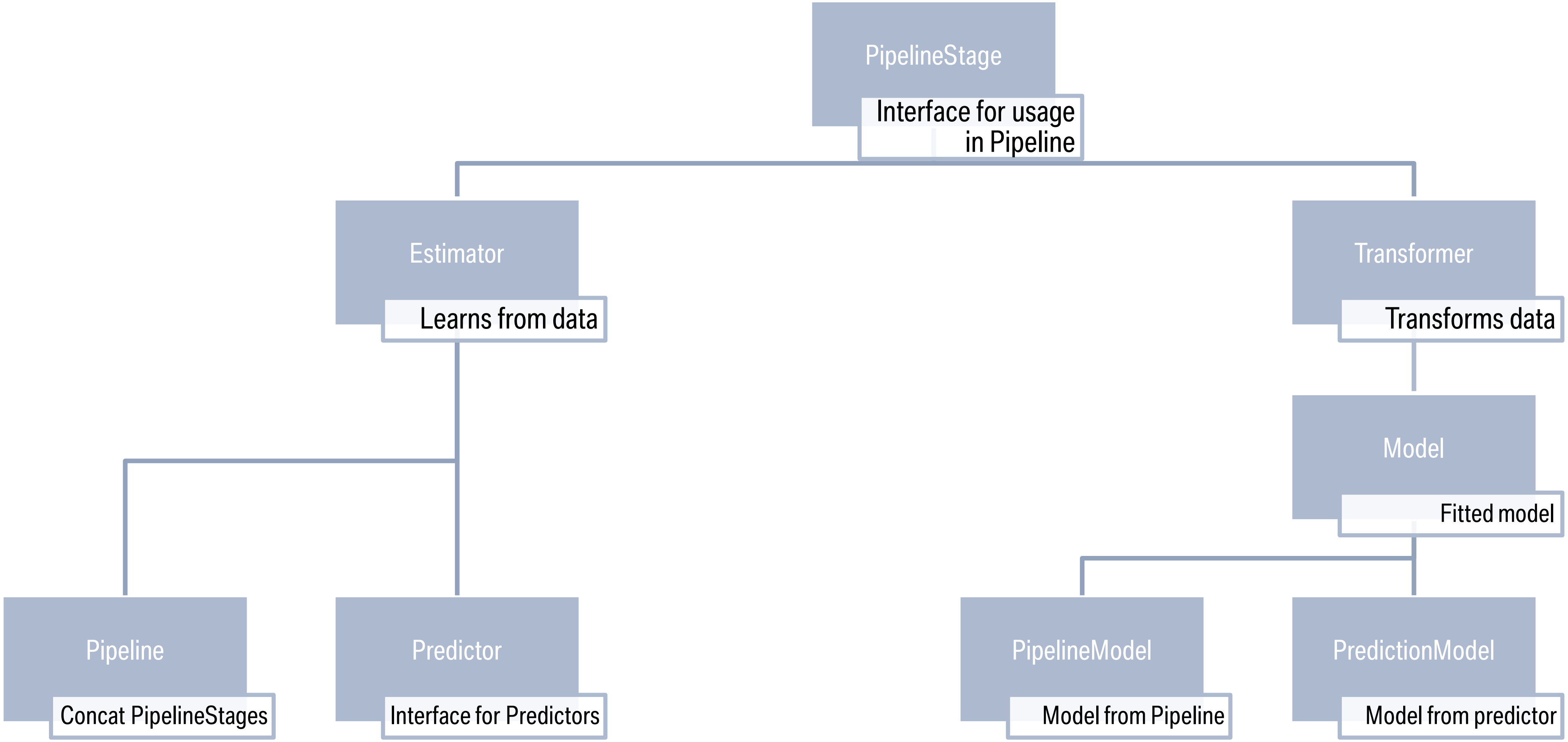
SPARK PIPELINE API.



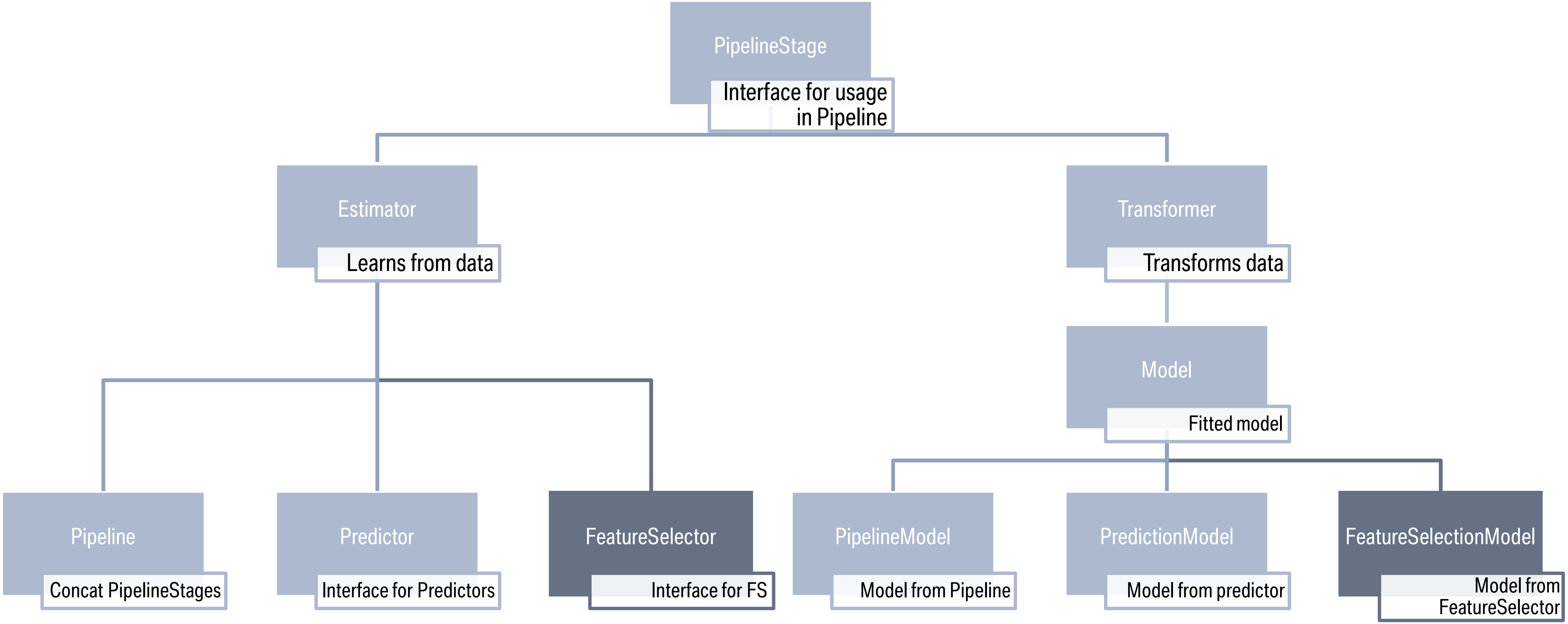
ORG.APACHE.SPARK.ML.*



ORG.APACHE.SPARK.ML.*



ORG.APACHE.SPARK.ML.*



MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

```scala
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
        M <: FeatureSelectorModel[M]]
           extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {









}
```

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {



}
```

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {



}
```

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {



}
```

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]

}
```

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

For setter concatenation:
mdl.setParam1(val1)
.setParam2(val2)...

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]









}
```

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

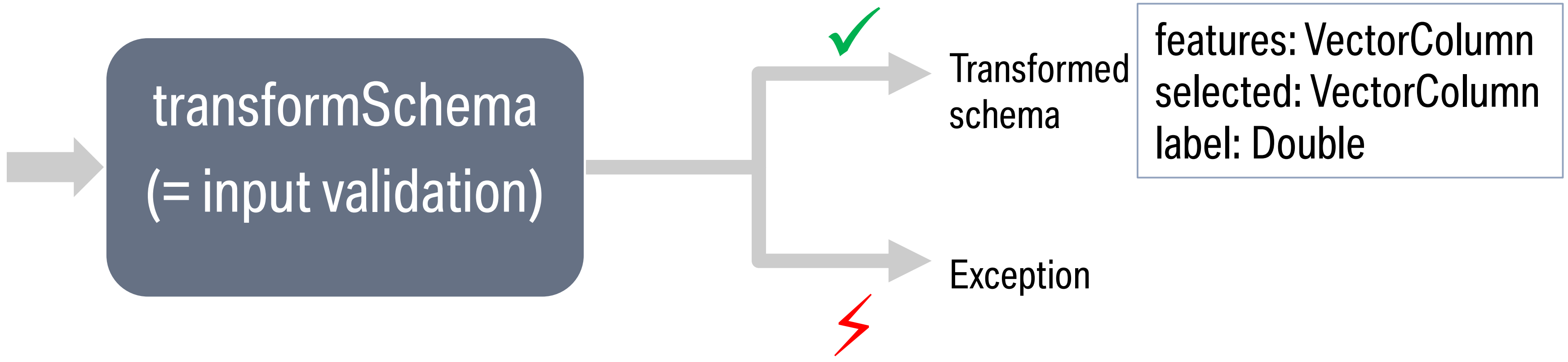
Makes all **Param** writable.

For setter concatenation:
mdl.setParam1(val1)
.setParam2(val2)...

```scala
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]
```

features	label
[0,1,0,1]	1.0
[1,0,0,0]	1.0

DataFrame with Schema




```scala
}
```

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

For setter concatenation:
mdl.setParam1(val1)
.setParam2(val2)...

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]
```

features	label
[0,1,0,1]	1.0
[1,0,0,0]	1.0

DataFrame with Schema

Attention:
VectorColumns have Metadata:
Name, Type, Range, etc.

```
}
```

transformSchema
(= input validation)

Transformed schema

features: VectorColumn
selected: VectorColumn
label: Double

Exception

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

For setter concatenation:
mdl.setParam1(val1)
.setParam2(val2)...

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
           extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]
  // PipelineStage and Estimator methods
  override def transformSchema(schema: StructType): StructType = {}




}
```

Performs input checking and fails fast. Can throw exceptions.

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

For setter concatenation: mdl.setParam1(val1)
.setParam2(val2)...

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]
  // PipelineStage and Estimator methods
  override def transformSchema(schema: StructType): StructType = {}
```

Performs input checking and fails fast. Can throw exceptions.

Dataset → **fit** (= learn from data) → Transformer

```
}
```

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

For setter concatenation:
mdl.setParam1(val1)
.setParam2(val2)...

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]
  // PipelineStage and Estimator methods
  override def transformSchema(schema: StructType): StructType = {}
  override def fit(dataset: Dataset[_]): M = {}
  override def copy(extra: ParamMap): Learner


}
```

Performs input checking and fails fast. Can throw exceptions.

Learns from data and returns a **Model**. Here: calculate feature importances.

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

For setter concatenation:
mdl.setParam1(val1)
.setParam2(val2)...

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
          extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]
  // PipelineStage and Estimator methods
  override def transformSchema(schema: StructType): StructType = {}
  override def fit(dataset: Dataset[_]): M = {}
  override def copy(extra: ParamMap): Learner
  // Abstract methods that are called from fit()
  protected def train(dataset: Dataset[_]): Array[(Int, Double)]
  protected def make(uid: String, selectedFeatures: Array[Int],
                     featureImportances: Map[String, Double]): M
}
```

Performs input checking and fails fast. Can throw exceptions.

Learns from data and returns a **Model**. Here: calculate feature importances.

MAKING AN ESTIMATOR – FEATURESELECTION EXAMPLE.

Needs to know, what it shall return.

Defined later.

Makes all **Param** writable.

For setter concatenation:
mdl.setParam1(val1)
.setParam2(val2)...

```
abstract class FeatureSelector[
  Learner <: FeatureSelector[Learner, M],
      M <: FeatureSelectorModel[M]]
           extends Estimator[M] with FeatureSelectorParams with DefaultParamsWritable {
  // Setters for params in FeatureSelectorParams
  def setParam*(value: ParamType): Learner = set(param, value).asInstanceOf[Learner]
  // PipelineStage and Estimator methods
  override def transformSchema(schema: StructType): StructType = {}
  override def fit(dataset: Dataset[_]): M = {}
  override def copy(extra: ParamMap): Learner
  // Abstract methods that are called from fit()
  protected def train(dataset: Dataset[_]): Array[(Int, Double)]
  protected def make(uid: String, selectedFeatures: Array[Int],
                     featureImportances: Map[String, Double]): M
}
```

Performs input checking and fails fast. Can throw exceptions.

Learns from data and returns a **Model**. Here: calculate feature importances.

Not necessary, but avoids code duplication.

MAKING A TRANSFORMER – FEATURESELECTION EXAMPLE.

```
abstract class FeatureSelectorModel[M <: FeatureSelectorModel[M]] (override val uid: String,
  val selectedFeatures: Array[Int],
  val featureImportances: Map[String, Double])
  extends Model[M] with FeatureSelectorParams with MLWritable{
```

MAKING A TRANSFORMER – FEATURESELECTION EXAMPLE.

```scala
abstract class FeatureSelectorModel[M <: FeatureSelectorModel[M]] (override val uid: String,
  val selectedFeatures: Array[Int],
  val featureImportances: Map[String, Double])
  extends Model[M] with FeatureSelectorParams with MLWritable{
```

For persistence.

MAKING A TRANSFORMER – FEATURESELECTION EXAMPLE.

```
abstract class FeatureSelectorModel[M <: FeatureSelectorModel[M]] (override val uid: String,
  val selectedFeatures: Array[Int],
  val featureImportances: Map[String, Double])
  extends Model[M] with FeatureSelectorParams with MLWritable{
  // Setters for params in FeatureSelectorParams
  def setFeaturesCol(value: String): this.type = set(featuresCol, value)

  // PipelineStage and Transformer methods
  override def transformSchema(schema: StructType): StructType = {}
  override def transform(dataset: Dataset[_]): DataFrame = {}
  def write: MLWriter
}
```

For persistence.

MAKING A TRANSFORMER – FEATURESELECTION EXAMPLE.

```
abstract class FeatureSelectorModel[M <: FeatureSelectorModel[M]] (override val uid: String,
  val selectedFeatures: Array[Int],
  val featureImportances: Map[String, Double])
  extends Model[M] with FeatureSelectorParams with MLWritable{
  // Setters for params in FeatureSelectorParams
  def setFeaturesCol(value: String): this.type = set(featuresCol, value)

  // PipelineStage and Transformer methods
  override def transformSchema(schema: StructType): StructType = {}
  override def transform(dataset: Dataset[_]): DataFrame = {}
  def write: MLWriter
}
```

For persistence.

Same idea as in Estimator, but different tasks.

MAKING A TRANSFORMER – FEATURESELECTION EXAMPLE.

```
abstract class FeatureSelectorModel[M <: FeatureSelectorModel[M]] (override val uid: String,
  val selectedFeatures: Array[Int],
  val featureImportances: Map[String, Double])
  extends Model[M] with FeatureSelectorParams with MLWritable{
  // Setters for params in FeatureSelectorParams
  def setFeaturesCol(value: String): this.type = set(featuresCol, value)

  // PipelineStage and Transformer methods
  override def transformSchema(schema: StructType): StructType = {}
  override def transform(dataset: Dataset[_]): DataFrame = {}
  def write: MLWriter
}
```

For persistence.

Same idea as in Estimator, but different tasks.

Transforms data.

MAKING A TRANSFORMER – FEATURESELECTION EXAMPLE.

```
abstract class FeatureSelectorModel[M <: FeatureSelectorModel[M]] (override val uid: String,
  val selectedFeatures: Array[Int],
  val featureImportances: Map[String, Double])
  extends Model[M] with FeatureSelectorParams with MLWritable{
  // Setters for params in FeatureSelectorParams
  def setFeaturesCol(value: String): this.type = set(featuresCol, value)

  // PipelineStage and Transformer methods
  override def transformSchema(schema: StructType): StructType = {}
  override def transform(dataset: Dataset[_]): DataFrame = {}
  def write: MLWriter
}
```

For persistence.

Same idea as in Estimator, but different tasks.

Transforms data.

Adds persistence.

GIVING YOUR NEW PIPELINESTAGE PARAMETERS.

```scala
import org.apache.spark.ml.param._
import org.apache.spark.ml.param.shared._

private[selection] trait FeatureSelectorParams extends Params
  with HasFeaturesCol with HasOutputCol with HasLabelCol {
  // Define params and getters here...
  final val param = new Param[Type](this, "name", "description")

  def getParam: Type = $(param)
}
```

GIVING YOUR NEW PIPELINESTAGE PARAMETERS.

Possible, because package is in org.apache.spark.ml.

```
import org.apache.spark.ml.param._
import org.apache.spark.ml.param.shared._

private[selection] trait FeatureSelectorParams extends Params
  with HasFeaturesCol with HasOutputCol with HasLabelCol {
  // Define params and getters here...
  final val param = new Param[Type](this, "name", "description")

  def getParam: Type = $(param)
}
```

GIVING YOUR NEW PIPELINESTAGE PARAMETERS.

```
import org.apache.spark.ml.param._
import org.apache.spark.ml.param.shared._

private[selection] trait FeatureSelectorParams extends Params
  with HasFeaturesCol with HasOutputCol with HasLabelCol {
  // Define params and getters here...
  final val param = new Param[Type](this, "name", "description")

  def getParam: Type = $(param)
}
```

Possible, because package is in org.apache.spark.ml.

Out of the box for several types, e.g.: DoubleParam, IntParam, BooleanParam, StringArrayParam,...

Other types: need to implement jsonEncode and jsonDecode to maintain persistence.

GIVING YOUR NEW PIPELINESTAGE PARAMETERS.

```scala
import org.apache.spark.ml.param._
import org.apache.spark.ml.param.shared._

private[selection] trait FeatureSelectorParams extends Params
  with HasFeaturesCol with HasOutputCol with HasLabelCol {
  // Define params and getters here...
  final val param = new Param[Type](this, "name", "description")

  def getParam: Type = $(param)
}
```

Possible, because package is in org.apache.spark.ml.

Out of the box for several types, e.g.: DoubleParam, IntParam, BooleanParam, StringArrayParam,...

Other types: need to implement jsonEncode and jsonDecode to maintain persistence.

getters are shared between Estimator and Transformer. setters not, for the pursuit of concatenation.

ADDING PERSISTENCE TO YOUR NEW PIPELINEMODEL.

- What has to be saved?
  - Metadata: uid, timestamp, version, …
  - Parameters
  - Learnt data: `selectedFeatures` & `featureImportances`

ADDING PERSISTENCE TO YOUR NEW PIPELINEMODEL.

- What has to be saved?
  - Metadata: uid, timestamp, version, …
  - Parameters
  - Learnt data: `selectedFeatures` & `featureImportances`

Since we are in org.apache.spark.ml, use:

```
DefaultParamsWriter.saveMetadata()
DefaultParamsReader.loadMetadata()
```

ADDING PERSISTENCE TO YOUR NEW PIPELINEMODEL.

- What has to be saved?
  - Metadata: uid, timestamp, version, …
  - Parameters
  - Learnt data: `selectedFeatures` & `featureImportances`

    → Create `DataFrame` and use `write.parquet(…)`

Since we are in org.apache.spark.ml, use:

```
DefaultParamsWriter.saveMetadata()
DefaultParamsReader.loadMetadata()
```

ADDING PERSISTENCE TO YOUR NEW PIPELINEMODEL.

- What has to be saved?
  - Metadata: uid, timestamp, version, …
  - Parameters

  Since we are in org.apache.spark.ml, use: `DefaultParamsWriter.saveMetadata()` `DefaultParamsReader.loadMetadata()`

  - Learnt data: `selectedFeatures` & `featureImportances`

    → Create `DataFrame` and use `write.parquet(…)`

- How do we do that?
  - Create companion `object FeatureSelectorModel`, which offers the following classes:
    - `abstract class FeatureSelectorModelReader[M <: FeatureSelectorModel[M]] extends MLReader[M] {…}`
    - `class FeatureSelectorModelWriter[M <: FeatureSelectorModel[M]](instance: M) extends MLWriter {…}`

HOW TO USE SPARK-FEATURESELECTION.

HOW TO USE SPARK-FEATURESELECTION.

```
import org.apache.spark.ml.feature.selection.filter._
import org.apache.spark.ml.feature.selection.util.VectorMerger
import org.apache.spark.ml.Pipeline
```

HOW TO USE SPARK-FEATURESELECTION.

df

features	Label
[0,1,0,1]	1.0
[0,0,0,0]	0.0
[1,1,0,0]	0.0
[1,0,0,0]	1.0

```
import org.apache.spark.ml.feature.selection.filter._
import org.apache.spark.ml.feature.selection.util.VectorMerger
import org.apache.spark.ml.Pipeline
```

```
// load Data
val df = spark.read.parquet("path/to/data/train.parquet")
```

HOW TO USE SPARK-FEATURESELECTION.

df

features	Label
[0,1,0,1]	1.0
[0,0,0,0]	0.0
[1,1,0,0]	0.0
[1,0,0,0]	1.0

```
import org.apache.spark.ml.feature.selection.filter._
import org.apache.spark.ml.feature.selection.util.VectorMerger
import org.apache.spark.ml.Pipeline
```

```
// load Data
val df = spark.read.parquet("path/to/data/train.parquet")
val corSel = new CorrelationSelector().setInputCol("features").setOutputCol(“cor")
val giniSel = new GiniSelector().setInputCol("features").setOutputCol(“gini")
```

Feature selectors. Offer different selection methods.

HOW TO USE SPARK-FEATURESELECTION.

df

features	Label
[0,1,0,1]	1.0
[0,0,0,0]	0.0
[1,1,0,0]	0.0
[1,0,0,0]	1.0

```
import org.apache.spark.ml.feature.selection.filter._
import org.apache.spark.ml.feature.selection.util.VectorMerger
import org.apache.spark.ml.Pipeline
```

```
// load Data
val df = spark.read.parquet("path/to/data/train.parquet")
val corSel = new CorrelationSelector().setInputCol("features").setOutputCol(“cor")
val giniSel = new GiniSelector().setInputCol("features").setOutputCol(“gini")
```

Feature selectors. Offer different selection methods.

```
// VectorMerger merges VectorColumns and removes duplicates. Requires vector columns with names!
val merger = new VectorMerger().setInputCols(Array(“cor", “gini")).setOutputCol(“selected")
```

HOW TO USE SPARK-FEATURESELECTION.

df

features	Label
[0,1,0,1]	1.0
[0,0,0,0]	0.0
[1,1,0,0]	0.0
[1,0,0,0]	1.0

fit

Feature	F1	F2	F3	F4
Score 1	0.9	0.7	0.0	0.5
Score 2	0.6	0.8	0.0	0.4

```
import org.apache.spark.ml.feature.selection.filter._
import org.apache.spark.ml.feature.selection.util.VectorMerger
import org.apache.spark.ml.Pipeline

// load Data
val df = spark.read.parquet("path/to/data/train.parquet")
val corSel = new CorrelationSelector().setInputCol("features").setOutputCol(“cor")
val giniSel = new GiniSelector().setInputCol("features").setOutputCol(“gini")

// VectorMerger merges VectorColumns and removes duplicates. Requires vector columns with names!
val merger = new VectorMerger().setInputCols(Array(“cor", “gini")).setOutputCol(“selected")
// Put everything in a pipeline and fit together
val plModel = new Pipeline().setStages(Array(corSel, giniSel, merger)).fit(df)
```

Feature selectors. Offer different selection methods.

HOW TO USE SPARK-FEATURESELECTION.

df

features	Label
[0,1,0,1]	1.0
[0,0,0,0]	0.0
[1,1,0,0]	0.0
[1,0,0,0]	1.0

dft

selected	Label
[0,1]	1.0
[0,0]	0.0
[1,1]	0.0
[1,0]	1.0

fit

Transform

Feature	F1	F2	F3	F4
Score 1	0.9	0.7	0.0	0.5
Score 2	0.6	0.8	0.0	0.4

```
import org.apache.spark.ml.feature.selection.filter._
import org.apache.spark.ml.feature.selection.util.VectorMerger
import org.apache.spark.ml.Pipeline
```

```
// load Data
val df = spark.read.parquet("path/to/data/train.parquet")
val corSel = new CorrelationSelector().setInputCol("features").setOutputCol("cor")
val giniSel = new GiniSelector().setInputCol("features").setOutputCol("gini")
```

Feature selectors. Offer different selection methods.

```
// VectorMerger merges VectorColumns and removes duplicates. Requires vector columns with names!
val merger = new VectorMerger().setInputCols(Array("cor", "gini")).setOutputCol("selected")
// Put everything in a pipeline and fit together
val plModel = new Pipeline().setStages(Array(corSel, giniSel, merger)).fit(df)
val dfT = plModel.transform(df).drop("Features")
```

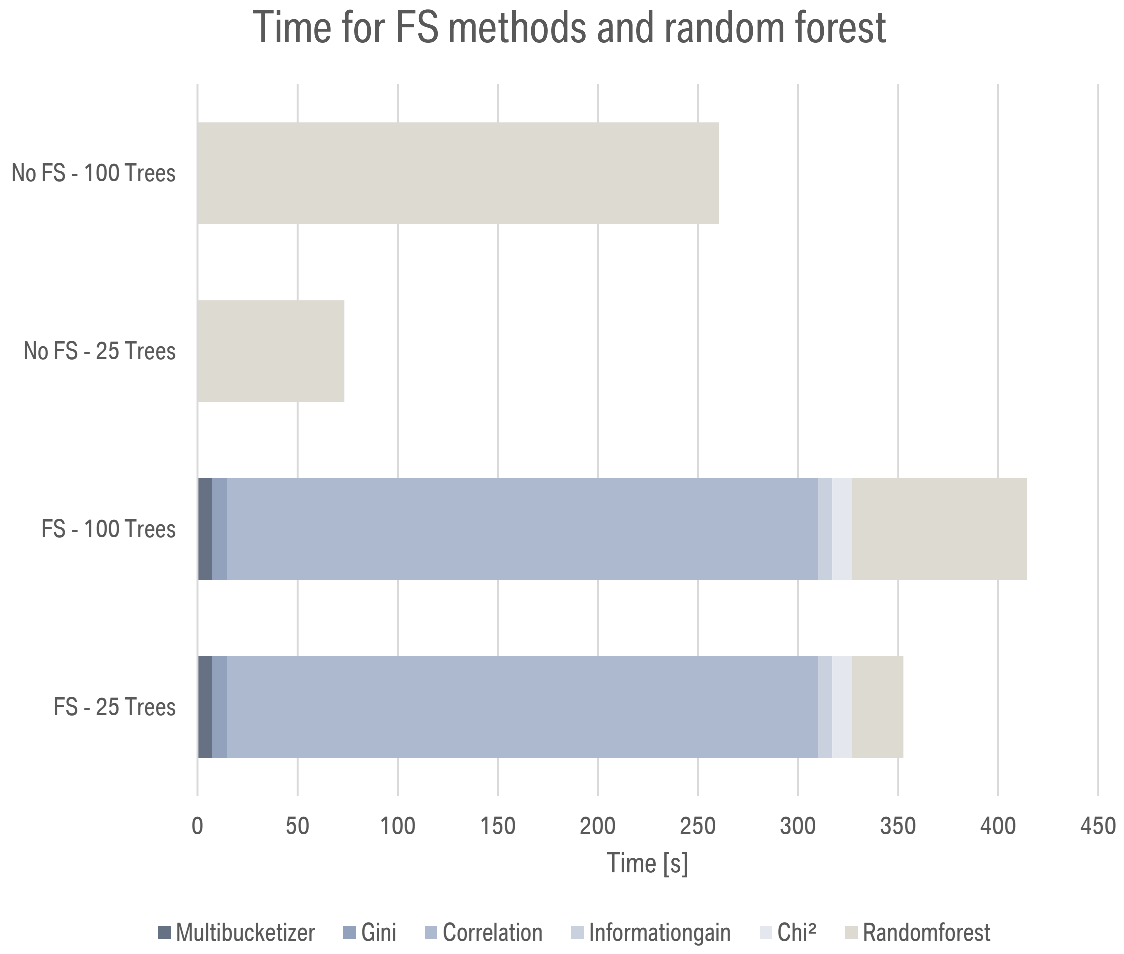
SPARK-FEATURESELECTION PACKAGE.

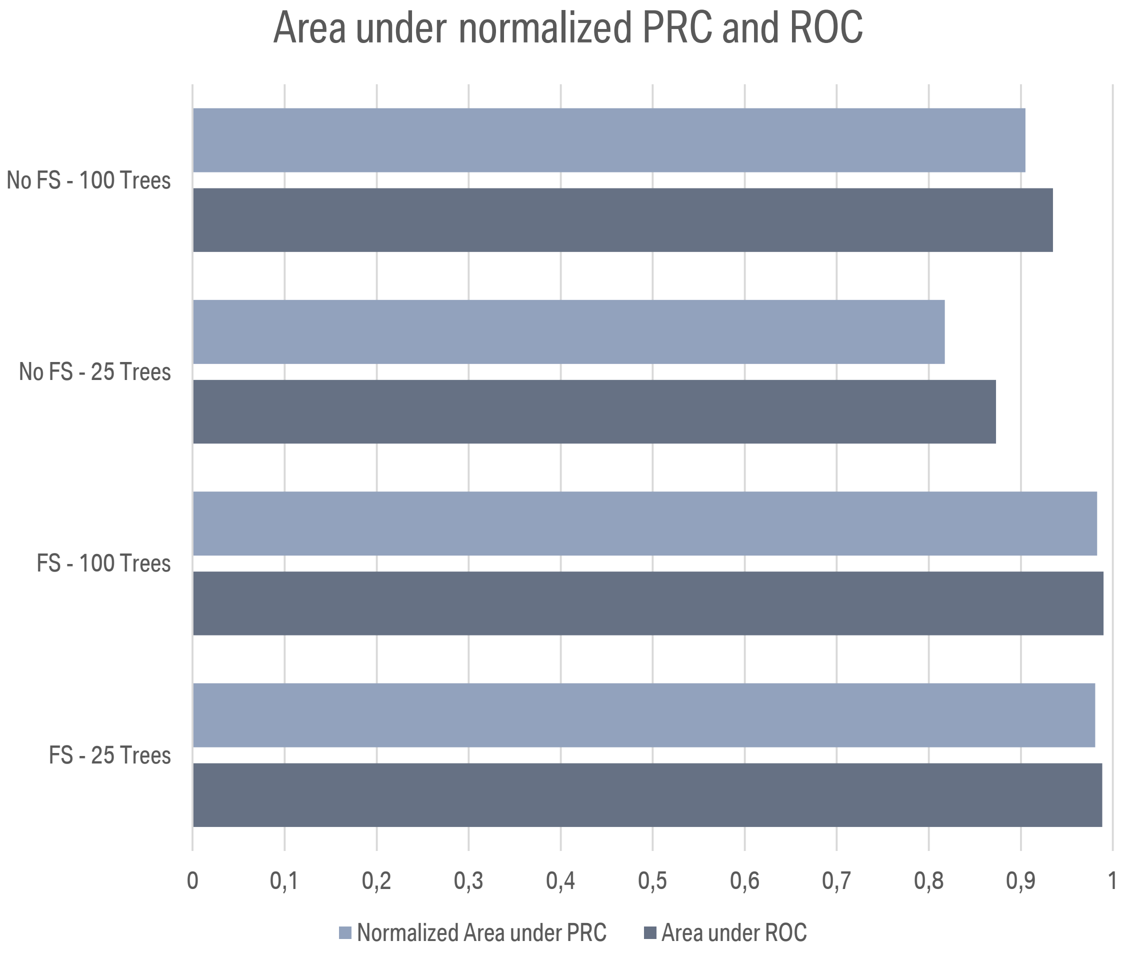
- Offers selection based on:
  - Gini coefficient
  - Correlation coefficient
  - Information gain
  - L1-Logistic regression weights
  - Randomforest importances
- Utility stage:
  - VectorMerger
- Three modes:
  - Percentile (default)
  - Fixed number of columns
  - Compare to random column [4]

Find on GitHub: spark-FeatureSelection or on Spark-packages

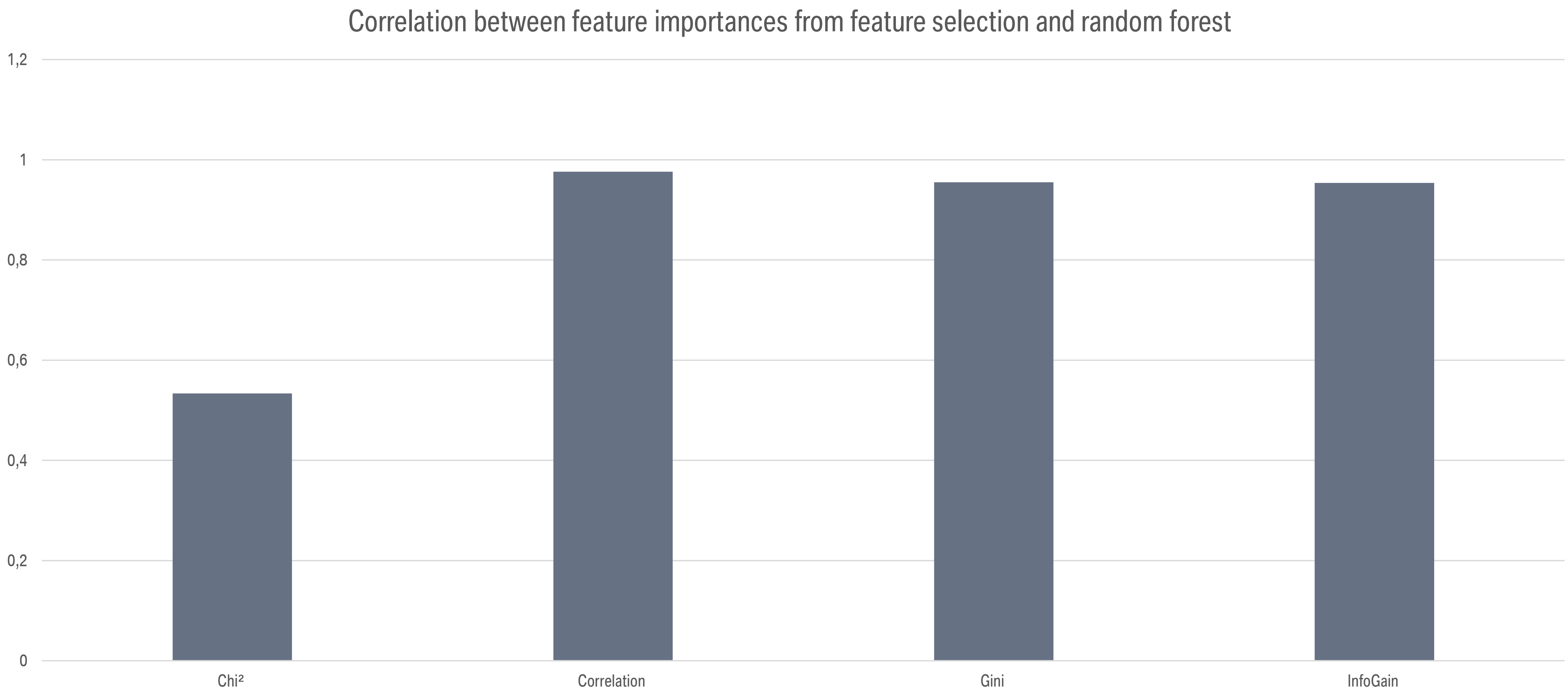
[4]: Stoppiglia et al.: Ranking a Random Feature for Variable and Feature Selection

PERFORMANCE.

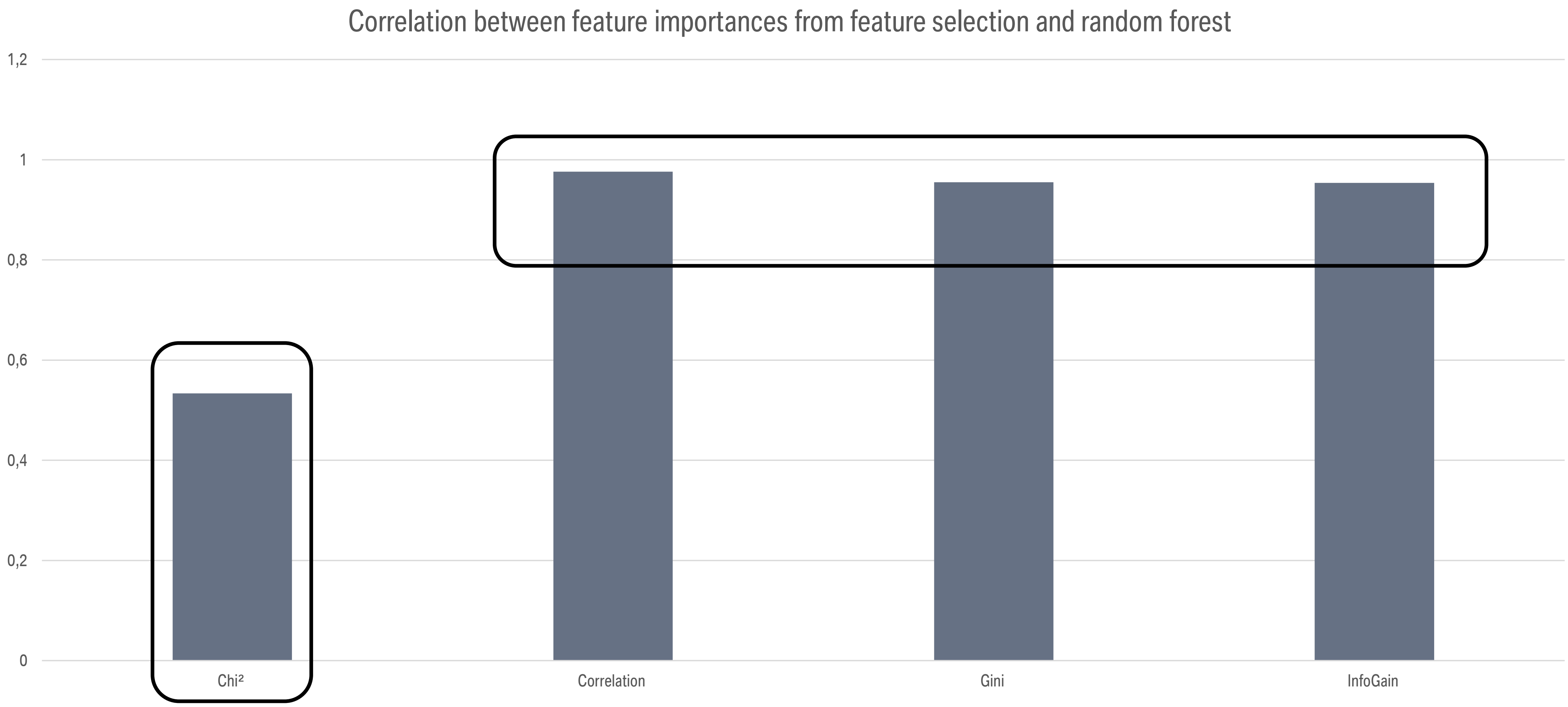




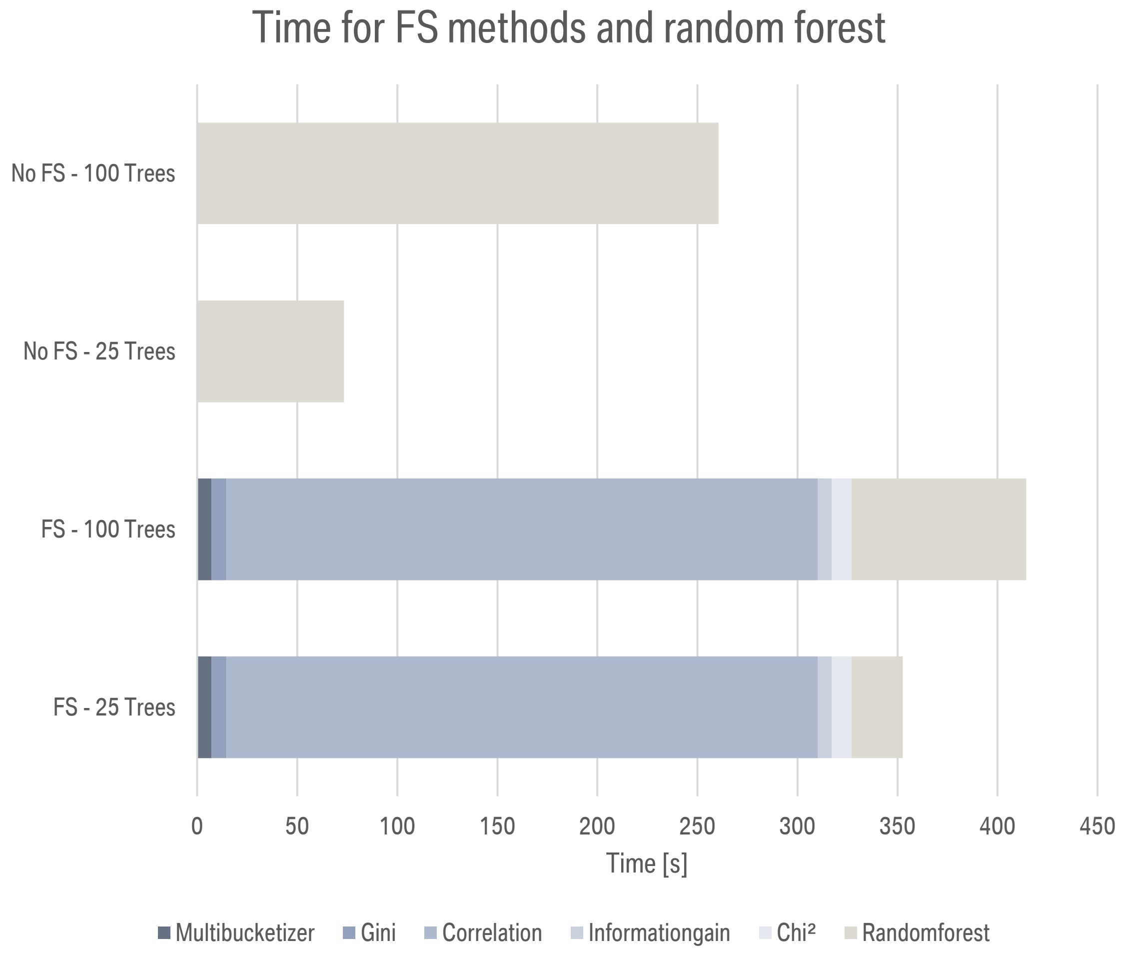
PERFORMANCE.

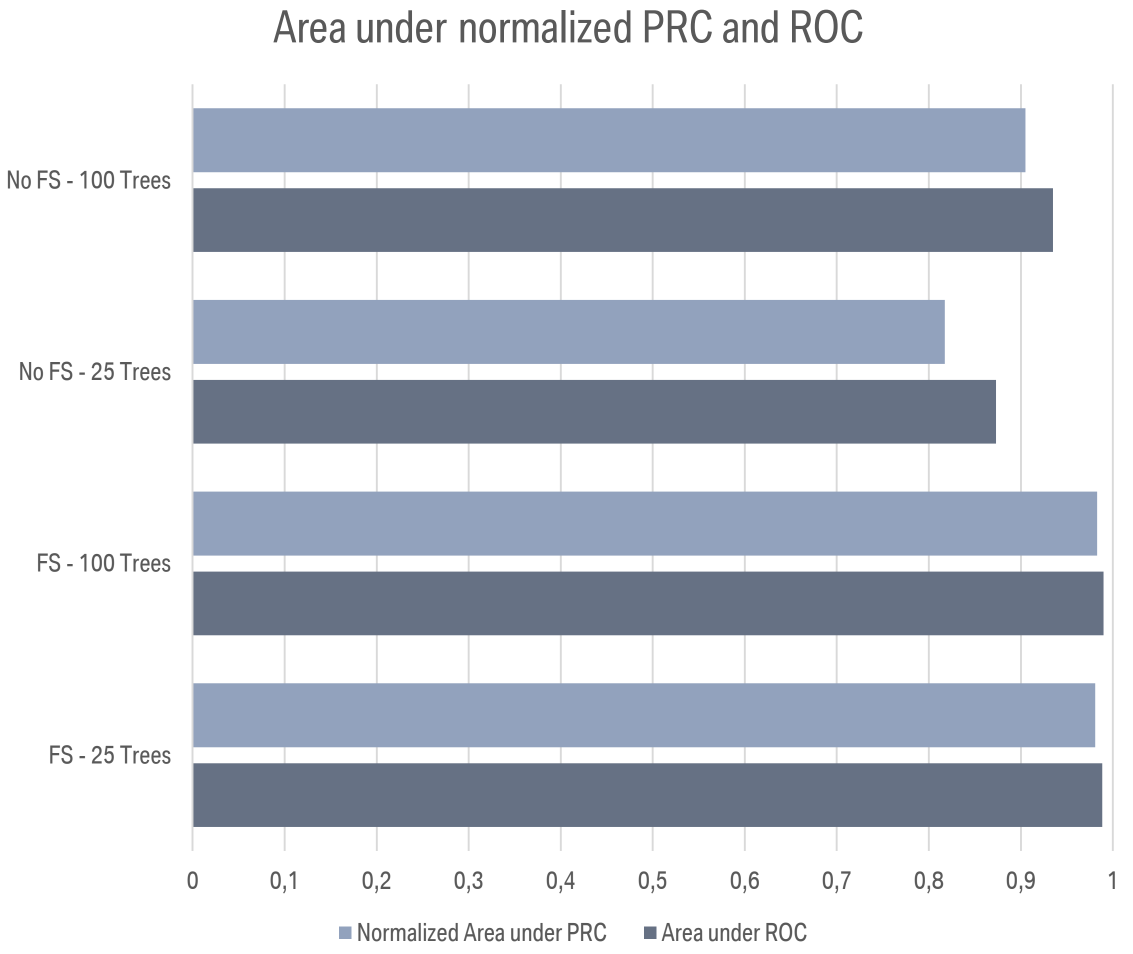


PERFORMANCE.



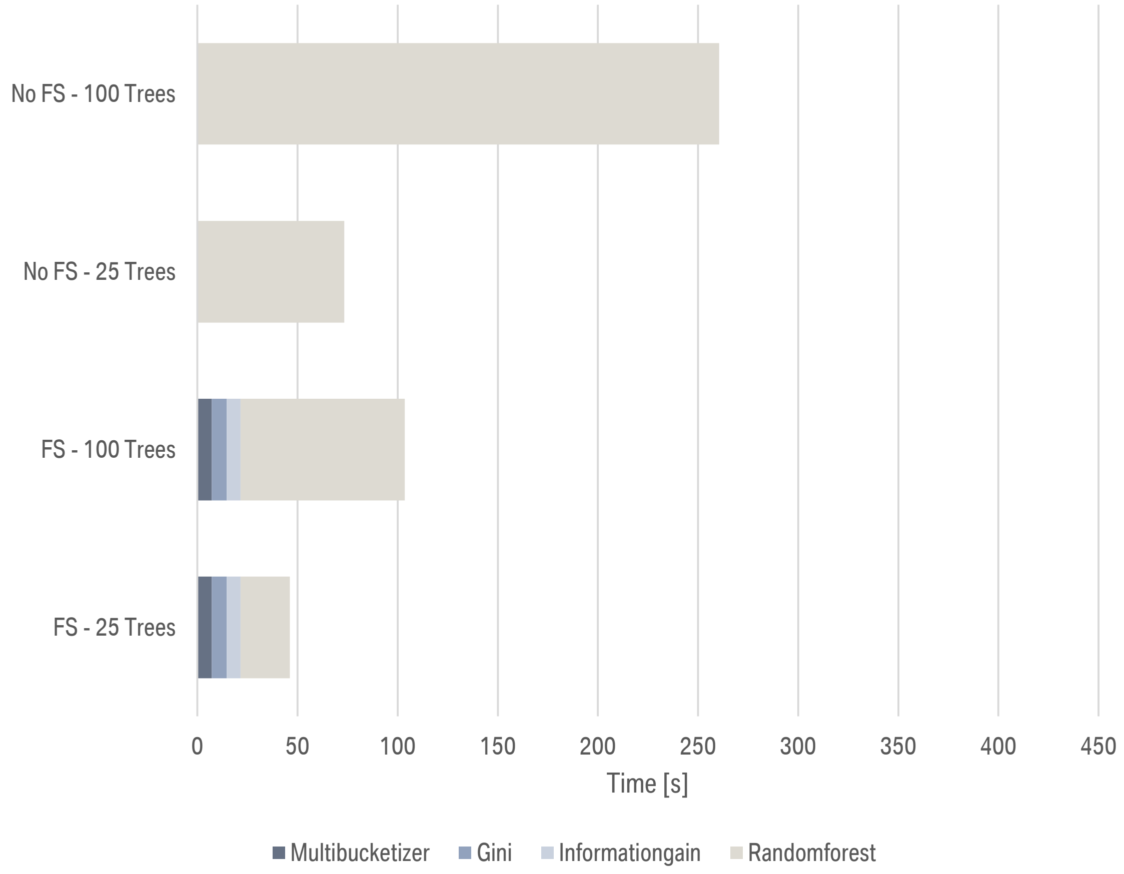
PERFORMANCE.



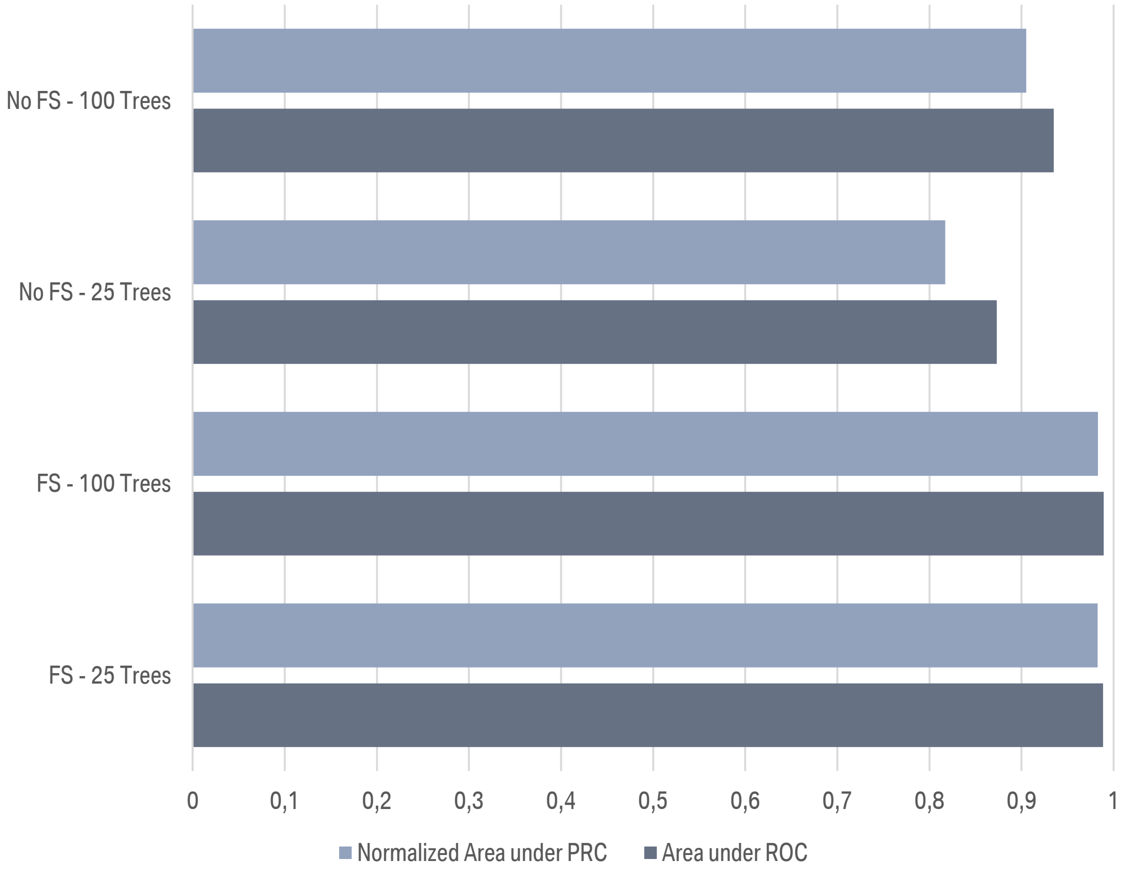


PERFORMANCE.

Time for FS methods and random forest



Area under normalized PRC and ROC



LESSONS LEARNT.

- Know what your data looks like and where it is located! Example:
  - Operations can succeed in local mode, but fail on a cluster.
  - Use .persist(`StorageLevel.MEMORY_ONLY`), when data fits into Memory. Default for `.cache` is `MEMORY_AND_DISK`.
- Do not reinvent the wheel for common methods → Consider putting your stages in to the spark.ml namespace.
- Use the Spark Web GUI to understand your Spark jobs.

QUESTIONS?

Marc.Kaminski@bmw.de
Bernhard.bb.Schegel@bmw.de

BACKUP.

DETERMINING WHERE YOUR PIPELINESTAGE SHOULD LIVE.

org.apache.spark.ml.*	
Pro	Con
Less code duplication (sharedParams, SchemaUtils, …)	More dangerous, when not cautious
Easier to implement persistence	

vs.

Own namespace	
Pro	Con
Safer solution	Code duplication

FEATURE SELECTION.

F1	F2	Noise	Label = F1 XOR F2
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

Feature Selection

Feature	Importance
Feature 1	0.7
Feature 2	0.7
Noise	0.2

F1	F2	Label = F1 XOR F2
0	0	0
1	0	1
0	1	1
1	1	0

- Motivation:
  - Many sparse features → feature space has to be reduced → select features that carry a lot of information for prediction.
  - Feature selection (unlike feature transformation) enables understanding of which features have a high impact on the model.

FEATURE SELECTION.

E.g.:
- Correlation
- InformationGain
- RandomForest
etc.

F1	F2	Noise	Label = F1 XOR F2
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

Feature Selection

Feature	Importance
Feature 1	0.7
Feature 2	0.7
Noise	0.2

F1	F2	Label = F1 XOR F2
0	0	0
1	0	1
0	1	1
1	1	0

- Motivation:
  - Many sparse features → feature space has to be reduced → select features that carry a lot of information for prediction.
  - Feature selection (unlike feature transformation) enables understanding of which features have a high impact on the model.

FEATURE SELECTION.

	Description	Advantages	Disadvantages	Examples
Filter	Evaluate intrinsic data properties	Fast Scalable	Ignore inter-feature dependencies Ignore interaction with classifier	Chi-squared Information gain Correlation
Wrapper	Evaluate model performance of feature subset	Feature dependencies Simple	Classifier dependent selection Computational expensive Risk of overfitting	Genetic algorithms Search algorithms
Embedded	Feature selection is embedded in classifier training	Feature dependencies	Classifier dependent selection	L1-Logistic regression Random forest

CHALLENGES.

– Big plans for DataFrames when performing many operations on many columns → Can take a long time to build and optimize DAG.
– Column limit for DataFrames introduced by several Jiras, especially: SPARK-18016 → Hopefully fixed in Spark 2.3.0.
– Spark PipelineStages are not consistent in how they handle DataFrame schemas → Sometimes no schema is appended.