pyspark.ml package

ML Pipeline APIs

DataFrame-based machine learning APIs to let users quickly assemble and configure practical machine learning pipelines.

class `pyspark.ml.`**`Transformer`** [source]

Abstract class for transformers that transform one dataset into another.

New in version 1.3.0.

`clear`(*param*)

Clears a param from the param map if it has been explicitly set.

`copy`(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. The default implementation creates a shallow copy using **`copy.copy()`**, and then copies the embedded and extra parameters over and returns the copy. Subclasses should override this method if the default approach is not sufficient.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

`explainParam`(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

`explainParams`()

Returns the documentation of all params with their optionally default values and user-supplied values.

`extractParamMap`(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

`getOrDefault`(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

`getParam`(*paramName*)

Gets a param by its name.

`hasDefault`(*param*)

Checks whether a param has a default value.

`hasParam`(*paramName*)

Tests whether this instance contains a param with a given (string) name.

`isDefined`(*param*)

Checks whether a param is explicitly set by user or has a default value.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

property **`params`**

Returns all params ordered by name. The default implementation uses **`dir()`** to get all attributes of type **`Param`**.

`set`(*param*, *value*)

Sets a parameter in the embedded param map.

`transform`(*dataset*, *params=None*) [source]

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **`pyspark.sql.DataFrame`**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

class `pyspark.ml.`**`UnaryTransformer`** [source]

Abstract class for transformers that take one input column, apply transformation, and output the result as a new column.

New in version 2.3.0.

`clear`(*param*)

Clears a param from the param map if it has been explicitly set.

`copy`(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. The default implementation creates a shallow copy using **`copy.copy()`**, and then copies the embedded and extra parameters over and returns the copy. Subclasses should override this method if the default approach is not sufficient.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

abstract **`createTransformFunc`**() [source]

Creates the transform function using the given param map. The input param map already takes account of the embedded param map. So the param values should be determined solely by the input param map.

`explainParam`(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

abstract **outputDataType**() [source]

Returns the data type of the output column.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

transformSchema(*schema*) [source]

abstract **validateInputType**(*inputType*) [source]

Validates the input type. Throw an exception if it is invalid.

class pyspark.ml.**Estimator** [source]

Abstract class for estimators that fit models to data.

New in version 1.3.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. The default implementation creates a shallow copy using **copy.copy()**, and then copies the embedded and extra parameters over and returns the copy. Subclasses should override this method if the default approach is not sufficient.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values

Returns:
merged param map

fit(*dataset*, *params=None*) [source]

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*) [source]

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

set(*param*, *value*)

Sets a parameter in the embedded param map.

class pyspark.ml.**Model** [source]

Abstract class for models that are fitted by estimators.

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. The default implementation creates a shallow copy using **copy.copy()**, and then copies the embedded and extra parameters over and returns the copy. Subclasses should override this method if the default approach is not sufficient.

Parameters:
extra – Extra parameters to copy to the new instance

Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values

Returns:
merged param map

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

set(*param*, *value*)

Sets a parameter in the embedded param map.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

class pyspark.ml.**Pipeline**(*stages=None*) [source]

A simple pipeline, which acts as an estimator. A Pipeline consists of a sequence of stages, each of which is either an **Estimator** or a **Transformer**. When **Pipeline.fit()** is called, the stages are executed in order. If a stage is an **Estimator**, its **Estimator.fit()** method will be called on the input dataset to fit a model. Then the model, which is a transformer, will be used to transform the dataset as the input to the next stage. If a stage is a **Transformer**, its **Transformer.transform()** method will be called to produce the dataset for the next stage. The fitted model from a **Pipeline** is a **PipelineModel**, which consists of fitted models and transformers, corresponding to the pipeline stages. If stages is an empty list, the pipeline acts as an identity transformer.

New in version 1.3.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*) [source]

Creates a copy of this instance.

Parameters:
extra – extra parameters
Returns:
new instance

New in version 1.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getStages() [source]

Get pipeline stages.

New in version 1.3.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**() [source]

Returns an MLReader instance for this class.

New in version 2.0.0.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setParams(*self*, *stages=None*) [source]

Sets params for Pipeline.

New in version 1.3.0.

setStages(*value*) [source]

Set pipeline stages.

Parameters:
value – a list of transformers or estimators
Returns:
the pipeline instance

New in version 1.3.0.

stages = *Param(parent='undefined', name='stages', doc='a list of pipeline stages')*

write() [source]

Returns an MLWriter instance for this ML instance.

New in version 2.0.0.

class pyspark.ml.**PipelineModel**(*stages*) [source]

Represents a compiled pipeline with transformers and fitted models.

New in version 1.3.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*) [source]

Creates a copy of this instance.

Parameters:
extra – extra parameters
Returns:
new instance

New in version 1.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**() [source]

Returns an MLReader instance for this class.

New in version 2.0.0.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write() [source]

Returns an MLWriter instance for this ML instance.

New in version 2.0.0.

pyspark.ml.param module

class pyspark.ml.param.**Param**(*parent*, *name*, *doc*, *typeConverter=None*) [source]

A param with self-contained documentation.

New in version 1.3.0.

class pyspark.ml.param.**Params** [source]

Components that take parameters. This also provides an internal param map to store parameter values attached to the instance.

New in version 1.3.0.

clear(*param*) [source]

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*) [source]

Creates a copy of this instance with the same uid and some extra params. The default implementation creates a shallow copy using **copy.copy()**, and then copies the embedded and extra parameters over and returns the copy. Subclasses should override this method if the default approach is not sufficient.

Parameters:
extra – Extra parameters to copy to the new instance

Returns:
Copy of this instance

explainParam(*param*) [source]

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams() [source]

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*) [source]

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values

Returns:
merged param map

getOrDefault(*param*) [source]

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*) [source]

Gets a param by its name.

hasDefault(*param*) [source]

Checks whether a param has a default value.

hasParam(*paramName*) [source]

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*) [source]

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*) [source]

Checks whether a param is explicitly set by user.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

set(*param*, *value*) [source]

Sets a parameter in the embedded param map.

class pyspark.ml.param.**TypeConverters** [source]

> **Note:** DeveloperApi

Factory methods for common type conversion functions for *Param.typeConverter*.

New in version 2.0.0.

static **identity**(*value*) [source]

Dummy converter that just returns value.

static **toBoolean**(*value*) [source]

Convert a value to a boolean, if possible.

static **toFloat**(*value*) [source]

Convert a value to a float, if possible.

static **toInt**(*value*) [source]

Convert a value to an int, if possible.

static **toList**(*value*) [source]

Convert a value to a list, if possible.

static **toListFloat**(*value*) [source]

Convert a value to list of floats, if possible.

static **toListInt**(*value*) [source]

Convert a value to list of ints, if possible.

static **toListListFloat**(*value*) [source]

Convert a value to list of list of floats, if possible.

static **toListString**(*value*) [source]

Convert a value to list of strings, if possible.

static **toMatrix**(*value*) [source]

Convert a value to a MLlib Matrix, if possible.

static **toString**(*value*) [source]

Convert a value to a string, if possible.

static **toVector**(*value*) [source]

Convert a value to a MLlib Vector, if possible.

pyspark.ml.feature module

class pyspark.ml.feature.**Binarizer**(*threshold=0.0*, *inputCol=None*, *outputCol=None*, *thresholds=None*, *inputCols=None*, *outputCols=None*) [source]

Binarize a column of continuous features given a threshold. Since 3.0.0, **Binarize** can map multiple columns at once by setting the **inputCols** parameter. Note that when both the **inputCol** and **inputCols** parameters are set, an Exception will be thrown. The **threshold** parameter is used for single column usage, and **thresholds** is for multiple columns.

```
>>> df = spark.createDataFrame([(0.5,)], ["values"])
>>> binarizer = Binarizer(threshold=1.0, inputCol="values", outputCol="features")
>>> binarizer.setThreshold(1.0)
Binarizer...
>>> binarizer.setInputCol("values")
Binarizer...
>>> binarizer.setOutputCol("features")
Binarizer...
>>> binarizer.transform(df).head().features
0.0
>>> binarizer.setParams(outputCol="freqs").transform(df).head().freqs
0.0
>>> params = {binarizer.threshold: -0.5, binarizer.outputCol: "vector"}
>>> binarizer.transform(df, params).head().vector
1.0
>>> binarizerPath = temp_path + "/binarizer"
>>> binarizer.save(binarizerPath)
>>> loadedBinarizer = Binarizer.load(binarizerPath)
>>> loadedBinarizer.getThreshold() == binarizer.getThreshold()
True
>>> df2 = spark.createDataFrame([(0.5, 0.3)], ["values1", "values2"])
>>> binarizer2 = Binarizer(thresholds=[0.0, 1.0])
>>> binarizer2.setInputCols(["values1", "values2"]).setOutputCols(["output1", "output2"])
Binarizer...
>>> binarizer2.transform(df2).show()
+-------+-------+-------+-------+
|values1|values2|output1|output2|
+-------+-------+-------+-------+
|    0.5|    0.3|    1.0|    0.0|
+-------+-------+-------+-------+
...
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getInputCols()

Gets the value of inputCols or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getOutputCols()
Gets the value of outputCols or its default value.

getParam(*paramName*)
Gets a param by its name.

getThreshold()
Gets the value of threshold or its default value.

getThresholds()
Gets the value of thresholds or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setInputCols(*value*) [source]
Sets the value of **inputCols**.

New in version 3.0.0.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

setOutputCols(*value*) [source]
Sets the value of **outputCols**.

New in version 3.0.0.

setParams(*self*, *threshold=0.0*, *inputCol=None*, *outputCol=None*, *thresholds=None*, *inputCols=None*, *outputCols=None*)
Sets params for this Binarizer. [source]

New in version 1.4.0.

setThreshold(*value*) [source]
Sets the value of **threshold**.

New in version 1.4.0.

setThresholds(*value*) [source]
Sets the value of **thresholds**.

New in version 3.0.0.

threshold = *Param(parent='undefined', name='threshold', doc='Param for threshold used to binarize continuous features. The features greater than the threshold will be binarized to 1.0. The features equal to or less than the threshold will be binarized to 0.0')*

thresholds = *Param(parent='undefined', name='thresholds', doc='Param for array of threshold used to binarize continuous features. This is for multiple columns input. If transforming multiple columns and thresholds is not set, but threshold is set, then threshold will be applied across all columns.')*

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**BucketedRandomProjectionLSH**(*inputCol=None*, *outputCol=None*, *seed=None*, *numHashTables=1*, *bucketLength=None*) [source]

LSH class for Euclidean distance metrics. The input is dense or sparse vectors, each of which represents a point in the Euclidean distance space. The output will be vectors of configurable dimension. Hash values in the same dimension are calculated by the same hash function.

See also: Stable Distributions

See also: Hashing for Similarity Search: A Survey

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.sql.functions import col
>>> data = [(0, Vectors.dense([-1.0, -1.0 ]),),
...         (1, Vectors.dense([-1.0, 1.0 ]),),
...         (2, Vectors.dense([1.0, -1.0 ]),),
...         (3, Vectors.dense([1.0, 1.0]),)]
>>> df = spark.createDataFrame(data, ["id", "features"])
>>> brp = BucketedRandomProjectionLSH()
>>> brp.setInputCol("features")
BucketedRandomProjectionLSH...
>>> brp.setOutputCol("hashes")
BucketedRandomProjectionLSH...
>>> brp.setSeed(12345)
BucketedRandomProjectionLSH...
>>> brp.setBucketLength(1.0)
BucketedRandomProjectionLSH...
>>> model = brp.fit(df)
>>> model.getBucketLength()
1.0
>>> model.setOutputCol("hashes")
BucketedRandomProjectionLSHModel...
>>> model.transform(df).head()
Row(id=0, features=DenseVector([-1.0, -1.0]), hashes=[DenseVector([-1.0])])
>>> data2 = [(4, Vectors.dense([2.0, 2.0 ]),),
...          (5, Vectors.dense([2.0, 3.0 ]),),
...          (6, Vectors.dense([3.0, 2.0 ]),),
...          (7, Vectors.dense([3.0, 3.0]),)]
>>> df2 = spark.createDataFrame(data2, ["id", "features"])
>>> model.approxNearestNeighbors(df2, Vectors.dense([1.0, 2.0]), 1).collect()
[Row(id=4, features=DenseVector([2.0, 2.0]), hashes=[DenseVector([1.0])], distCol=1.0)]
>>> model.approxSimilarityJoin(df, df2, 3.0, distCol="EuclideanDistance").select(
...     col("datasetA.id").alias("idA"),
...     col("datasetB.id").alias("idB"),
...     col("EuclideanDistance")).show()
+---+---+-----------------+
|idA|idB|EuclideanDistance|
+---+---+-----------------+
|  3|  6| 2.23606797749979|
+---+---+-----------------+
...
>>> model.approxSimilarityJoin(df, df2, 3, distCol="EuclideanDistance").select(
...     col("datasetA.id").alias("idA"),
...     col("datasetB.id").alias("idB"),
...     col("EuclideanDistance")).show()
+---+---+-----------------+
|idA|idB|EuclideanDistance|
+---+---+-----------------+
|  3|  6| 2.23606797749979|
+---+---+-----------------+
...
>>> brpPath = temp_path + "/brp"
>>> brp.save(brpPath)
>>> brp2 = BucketedRandomProjectionLSH.load(brpPath)
>>> brp2.getBucketLength() == brp.getBucketLength()
True
>>> modelPath = temp_path + "/brp-model"
>>> model.save(modelPath)
>>> model2 = BucketedRandomProjectionLSHModel.load(modelPath)
>>> model.transform(df).head().hashes == model2.transform(df).head().hashes
True
```

New in version 2.2.0.

bucketLength = *Param(parent='undefined', name='bucketLength', doc='the length of each hash bucket, a larger bucket lowers the false negative rate.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:

A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getBucketLength()

Gets the value of bucketLength or its default value.

New in version 2.2.0.

getInputCol()

Gets the value of inputCol or its default value.

getNumHashTables()

Gets the value of numHashTables or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

getSeed()

Gets the value of seed or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numHashTables = *Param(parent='undefined', name='numHashTables', doc='number of hash tables, where increasing number of hash tables lowers the false negative rate, and decreasing it improves the running performance.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setBucketLength(*value*) [source]

Sets the value of **bucketLength**.

New in version 2.2.0.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setNumHashTables(*value*)

Sets the value of **numHashTables**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *inputCol=None*, *outputCol=None*, *seed=None*, *numHashTables=1*, *bucketLength=None*) [source]

Sets params for this BucketedRandomProjectionLSH.

New in version 2.2.0.

setSeed(*value*) [source]

Sets the value of **seed**.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**BucketedRandomProjectionLSHModel**(*java_model=None*) [source]

Model fitted by **BucketedRandomProjectionLSH**, where multiple random vectors are stored. The vectors are normalized to be unit vectors and each vector is used in a hash function: \(h_i(x) = floor(r_i \cdot x / bucketLength)\) where \(r_i\) is the i-th random unit vector. The number of buckets will be *(max L2 norm of input vectors) / bucketLength*.

New in version 2.2.0.

approxNearestNeighbors(*dataset*, *key*, *numNearestNeighbors*, *distCol='distCol'*)

Given a large dataset and an item, approximately find at most k items which have the closest distance to the item. If the **outputCol** is missing, the method will transform the data; if the **outputCol** exists, it will use that. This allows caching of the transformed data when necessary.

> **Note:** This method is experimental and will likely change behavior in the next release.

Parameters:
- **dataset** – The dataset to search for nearest neighbors of the key.
- **key** – Feature vector representing the item to search for.
- **numNearestNeighbors** – The maximum number of nearest neighbors.
- **distCol** – Output column for storing the distance between each result row and the key. Use “distCol” as default value if it’s not specified.

Returns:
A dataset containing at most k items closest to the key. A column “distCol” is added to show the distance between each row and the key.

approxSimilarityJoin(*datasetA*, *datasetB*, *threshold*, *distCol='distCol'*)

Join two datasets to approximately find all pairs of rows whose distance are smaller than the threshold. If the **outputCol** is missing, the method will transform the data; if the **outputCol** exists, it will use that. This allows caching of the transformed data when necessary.

Parameters:
- **datasetA** – One of the datasets to join.
- **datasetB** – Another dataset to join.
- **threshold** – The threshold for the distance of row pairs.
- **distCol** – Output column for storing the distance between each pair of rows. Use “distCol” as default value if it’s not specified.

Returns:
A joined dataset containing pairs of rows. The original rows are in columns “datasetA” and “datasetB”, and a column “distCol” is added to show the distance between each pair.

bucketLength = *Param(parent='undefined', name='bucketLength', doc='the length of each hash bucket, a larger bucket lowers the false negative rate.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getBucketLength()

Gets the value of bucketLength or its default value.

New in version 2.2.0.

getInputCol()

Gets the value of inputCol or its default value.

getNumHashTables()

Gets the value of numHashTables or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numHashTables = *Param(parent='undefined', name='numHashTables', doc='number of hash tables, where increasing number of hash tables lowers the false negative rate, and decreasing it improves the running performance.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**Bucketizer**(*splits=None*, *inputCol=None*, *outputCol=None*, *handleInvalid='error'*, *splitsArray=None*, *inputCols=None*, *outputCols=None*) [source]

Maps a column of continuous features to a column of feature buckets. Since 3.0.0, **Bucketizer** can map multiple columns at once by setting the **inputCols** parameter. Note that when both the **inputCol** and **inputCols** parameters are set, an Exception will be thrown. The **splits** parameter is only used for single column usage, and **splitsArray** is for multiple columns.

```
>>> values = [(0.1, 0.0), (0.4, 1.0), (1.2, 1.3), (1.5, float("nan")),
...     (float("nan"), 1.0), (float("nan"), 0.0)]
>>> df = spark.createDataFrame(values, ["values1", "values2"])
>>> bucketizer = Bucketizer()
>>> bucketizer.setSplits([-float("inf"), 0.5, 1.4, float("inf")])
Bucketizer...
>>> bucketizer.setInputCol("values1")
Bucketizer...
>>> bucketizer.setOutputCol("buckets")
Bucketizer...
>>> bucketed = bucketizer.setHandleInvalid("keep").transform(df).collect()
>>> bucketed = bucketizer.setHandleInvalid("keep").transform(df.select("values1"))
>>> bucketed.show(truncate=False)
+-------+-------+
|values1|buckets|
+-------+-------+
|0.1    |0.0    |
|0.4    |0.0    |
|1.2    |1.0    |
|1.5    |2.0    |
|NaN    |3.0    |
|NaN    |3.0    |
+-------+-------+
...
>>> bucketizer.setParams(outputCol="b").transform(df).head().b
0.0
>>> bucketizerPath = temp_path + "/bucketizer"
>>> bucketizer.save(bucketizerPath)
>>> loadedBucketizer = Bucketizer.load(bucketizerPath)
>>> loadedBucketizer.getSplits() == bucketizer.getSplits()
True
>>> bucketed = bucketizer.setHandleInvalid("skip").transform(df).collect()
>>> len(bucketed)
4
>>> bucketizer2 = Bucketizer(splitsArray=
...     [[-float("inf"), 0.5, 1.4, float("inf")], [-float("inf"), 0.5, float("inf")]],
...     inputCols=["values1", "values2"], outputCols=["buckets1", "buckets2"])
>>> bucketed2 = bucketizer2.setHandleInvalid("keep").transform(df)
>>> bucketed2.show(truncate=False)
+-------+-------+--------+--------+
|values1|values2|buckets1|buckets2|
+-------+-------+--------+--------+
|0.1    |0.0    |0.0     |0.0     |
|0.4    |1.0    |0.0     |1.0     |
|1.2    |1.3    |1.0     |1.0     |
|1.5    |NaN    |2.0     |2.0     |
|NaN    |1.0    |3.0     |1.0     |
|NaN    |0.0    |3.0     |0.0     |
+-------+-------+--------+--------+
...
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getInputCol()

Gets the value of inputCol or its default value.

getInputCols()

Gets the value of inputCols or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getOutputCols()

Gets the value of outputCols or its default value.

getParam(*paramName*)

Gets a param by its name.

getSplits() [source]

Gets the value of threshold or its default value.

New in version 1.4.0.

getSplitsArray() [source]

Gets the array of split points or its default value.

New in version 3.0.0.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="how to handle invalid entries containing NaN values. Values outside the splits will always be treated as errors. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (keep invalid values in a special additional bucket). Note that in the multiple column case, the invalid handling is applied to all columns. That said for 'error' it will throw an error if any invalids are found in any column, for 'skip' it will skip rows with any invalids in any columns, etc.")*

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setHandleInvalid(*value*) [source]

Sets the value of **handleInvalid**.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setInputCols(*value*) [source]

Sets the value of **inputCols**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setOutputCols(*value*) [source]

Sets the value of **outputCols**.

New in version 3.0.0.

setParams(*self*, *splits=None*, *inputCol=None*, *outputCol=None*, *handleInvalid="error"*, *splitsArray=None*, *inputCols=None*, *outputCols=None*) [source]

Sets params for this Bucketizer.

New in version 1.4.0.

setSplits(*value*) [source]

Sets the value of **splits**.

New in version 1.4.0.

setSplitsArray(*value*) [source]

Sets the value of **splitsArray**.

New in version 3.0.0.

splits = *Param(parent='undefined', name='splits', doc='Split points for mapping continuous features into buckets. With n+1 splits, there are n buckets. A bucket defined by splits x,y holds values in the range [x,y) except the last bucket, which also includes y. The splits should be of length >= 3 and strictly increasing. Values at -inf, inf must be explicitly provided to cover all Double values; otherwise, values outside the splits specified will be treated as errors.')*

splitsArray = *Param(parent='undefined', name='splitsArray', doc='The array of split points for mapping continuous features into buckets for multiple columns. For each input column, with n+1 splits, there are n buckets. A bucket defined by splits x,y holds values in the range [x,y) except the last bucket, which also includes y. The splits should be of length >= 3 and strictly increasing. Values at -inf, inf must be explicitly provided to cover all Double values; otherwise, values outside the splits specified will be treated as errors.')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**ChiSqSelector**(*numTopFeatures=50*, *featuresCol='features'*, *outputCol=None*, *labelCol='label'*, *selectorType='numTopFeatures'*, *percentile=0.1*, *fpr=0.05*, *fdr=0.05*, *fwe=0.05*) [source]

Chi-Squared feature selection, which selects categorical features to use for predicting a categorical label. The selector supports different selection methods: *numTopFeatures*, *percentile*, *fpr*, *fdr*, *fwe*.

- *numTopFeatures* chooses a fixed number of top features according to a chi-squared test.
- *percentile* is similar but chooses a fraction of all features instead of a fixed number.
- *fpr* chooses all features whose p-values are below a threshold, thus controlling the false positive rate of selection.
- *fdr* uses the Benjamini-Hochberg procedure to choose all features whose false discovery rate is below a threshold.
- *fwe* chooses all features whose p-values are below a threshold. The threshold is scaled by 1/numFeatures, thus controlling the family-wise error rate of selection.

By default, the selection method is *numTopFeatures*, with the default number of top features set to 50.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame(
...    [(Vectors.dense([0.0, 0.0, 18.0, 1.0]), 1.0),
...     (Vectors.dense([0.0, 1.0, 12.0, 0.0]), 0.0),
...     (Vectors.dense([1.0, 0.0, 15.0, 0.1]), 0.0)],
...    ["features", "label"])
>>> selector = ChiSqSelector(numTopFeatures=1, outputCol="selectedFeatures")
>>> model = selector.fit(df)
>>> model.getFeaturesCol()
'features'
>>> model.setFeaturesCol("features")
ChiSqSelectorModel...
>>> model.transform(df).head().selectedFeatures
DenseVector([18.0])
>>> model.selectedFeatures
[2]
>>> chiSqSelectorPath = temp_path + "/chi-sq-selector"
>>> selector.save(chiSqSelectorPath)
>>> loadedSelector = ChiSqSelector.load(chiSqSelectorPath)
>>> loadedSelector.getNumTopFeatures() == selector.getNumTopFeatures()
True
>>> modelPath = temp_path + "/chi-sq-selector-model"
>>> model.save(modelPath)
>>> loadedModel = ChiSqSelectorModel.load(modelPath)
>>> loadedModel.selectedFeatures == model.selectedFeatures
True
```

New in version 2.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance

Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from

input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fdr = *Param(parent='undefined', name='fdr', doc='The upper bound of the expected false discovery rate.')*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

fpr = *Param(parent='undefined', name='fpr', doc='The highest p-value for features to be kept.')*

fwe = *Param(parent='undefined', name='fwe', doc='The upper bound of the expected family-wise error rate.')*

getFdr()

Gets the value of fdr or its default value.

New in version 2.2.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getFpr()

Gets the value of fpr or its default value.

New in version 2.1.0.

getFwe()

Gets the value of fwe or its default value.

New in version 2.2.0.

getLabelCol()

Gets the value of labelCol or its default value.

getNumTopFeatures()

Gets the value of numTopFeatures or its default value.

New in version 2.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

getPercentile()

Gets the value of percentile or its default value.

New in version 2.1.0.

getSelectorType()

Gets the value of selectorType or its default value.

New in version 2.1.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numTopFeatures = *Param(parent='undefined', name='numTopFeatures', doc='Number of features that selector will select,*

ordered by ascending p-value. If the number of features is < numTopFeatures, then this will select all features.')

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

percentile = *Param(parent='undefined', name='percentile', doc='Percentile of features that selector will select, ordered by ascending p-value.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

selectorType = *Param(parent='undefined', name='selectorType', doc='The selector type of the ChisqSelector. Supported options: numTopFeatures (default), percentile, fpr, fdr, fwe.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFdr(*value*) [source]

Sets the value of **fdr**. Only applicable when selectorType = "fdr".

New in version 2.2.0.

setFeaturesCol(*value*) [source]

Sets the value of **featuresCol**.

setFpr(*value*) [source]

Sets the value of **fpr**. Only applicable when selectorType = "fpr".

New in version 2.1.0.

setFwe(*value*) [source]

Sets the value of **fwe**. Only applicable when selectorType = "fwe".

New in version 2.2.0.

setLabelCol(*value*) [source]

Sets the value of **labelCol**.

setNumTopFeatures(*value*) [source]

Sets the value of **numTopFeatures**. Only applicable when selectorType = "numTopFeatures".

New in version 2.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *numTopFeatures=50*, *featuresCol="features"*, *outputCol=None*, *labelCol="labels"*, *selectorType="numTopFeatures"*, *percentile=0.1*, *fpr=0.05*, *fdr=0.05*, *fwe=0.05*) [source]

Sets params for this ChiSqSelector.

New in version 2.0.0.

setPercentile(*value*) [source]

Sets the value of **percentile**. Only applicable when selectorType = "percentile".

New in version 2.1.0.

setSelectorType(*value*) [source]

Sets the value of **selectorType**.

New in version 2.1.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**ChiSqSelectorModel**(*java_model=None*) [source]

Model fitted by **ChiSqSelector**.

New in version 2.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fdr = *Param(parent='undefined', name='fdr', doc='The upper bound of the expected false discovery rate.')*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fpr = *Param(parent='undefined', name='fpr', doc='The highest p-value for features to be kept.')*

fwe = *Param(parent='undefined', name='fwe', doc='The upper bound of the expected family-wise error rate.')*

getFdr()
Gets the value of fdr or its default value.

New in version 2.2.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getFpr()
Gets the value of fpr or its default value.

New in version 2.1.0.

getFwe()
Gets the value of fwe or its default value.

New in version 2.2.0.

getLabelCol()
Gets the value of labelCol or its default value.

getNumTopFeatures()
Gets the value of numTopFeatures or its default value.

New in version 2.0.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

getPercentile()
Gets the value of percentile or its default value.

New in version 2.1.0.

getSelectorType()
Gets the value of selectorType or its default value.

New in version 2.1.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numTopFeatures = *Param(parent='undefined', name='numTopFeatures', doc='Number of features that selector will select, ordered by ascending p-value. If the number of features is < numTopFeatures, then this will select all features.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

percentile = *Param(parent='undefined', name='percentile', doc='Percentile of features that selector will select, ordered by ascending p-value.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

property **selectedFeatures**
List of indices to select (filter).

New in version 2.0.0.

selectorType = *Param(parent='undefined', name='selectorType', doc='The selector type of the ChisqSelector. Supported options: numTopFeatures (default), percentile, fpr, fdr, fwe.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

New in version 3.0.0.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**CountVectorizer**(*minTF=1.0*, *minDF=1.0*, *maxDF=9223372036854775807*, *vocabSize=262144*, *binary=False*, *inputCol=None*, *outputCol=None*) [source]

Extracts a vocabulary from document collections and generates a **CountVectorizerModel**.

```
>>> df = spark.createDataFrame(
...    [(0, ["a", "b", "c"]), (1, ["a", "b", "b", "c", "a"])],
...    ["label", "raw"])
>>> cv = CountVectorizer()
>>> cv.setInputCol("raw")
CountVectorizer...
>>> cv.setOutputCol("vectors")
CountVectorizer...
>>> model = cv.fit(df)
>>> model.setInputCol("raw")
CountVectorizerModel...
>>> model.transform(df).show(truncate=False)
+-----+---------------+-------------------------+
|label|raw            |vectors                  |
+-----+---------------+-------------------------+
|0    |[a, b, c]      |(3,[0,1,2],[1.0,1.0,1.0])|
|1    |[a, b, b, c, a]|(3,[0,1,2],[2.0,2.0,1.0])|
+-----+---------------+-------------------------+
...
>>> sorted(model.vocabulary) == ['a', 'b', 'c']
True
>>> countVectorizerPath = temp_path + "/count-vectorizer"
>>> cv.save(countVectorizerPath)
>>> loadedCv = CountVectorizer.load(countVectorizerPath)
>>> loadedCv.getMinDF() == cv.getMinDF()
True
>>> loadedCv.getMinTF() == cv.getMinTF()
True
>>> loadedCv.getVocabSize() == cv.getVocabSize()
True
>>> modelPath = temp_path + "/count-vectorizer-model"
>>> model.save(modelPath)
>>> loadedModel = CountVectorizerModel.load(modelPath)
>>> loadedModel.vocabulary == model.vocabulary
True
>>> fromVocabModel = CountVectorizerModel.from_vocabulary(["a", "b", "c"],
...     inputCol="raw", outputCol="vectors")
>>> fromVocabModel.transform(df).show(truncate=False)
+-----+---------------+-------------------------+
|label|raw            |vectors                  |
+-----+---------------+-------------------------+
|0    |[a, b, c]      |(3,[0,1,2],[1.0,1.0,1.0])|
|1    |[a, b, b, c, a]|(3,[0,1,2],[2.0,2.0,1.0])|
+-----+---------------+-------------------------+
...
```

New in version 1.6.0.

binary = *Param(parent='undefined', name='binary', doc='Binary toggle to control the output vector values. If True, all nonzero counts (after minTF filter applied) are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. Default False')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of `pyspark.sql.DataFrame`.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getBinary()
Gets the value of binary or its default value.

New in version 2.0.0.

getInputCol()
Gets the value of inputCol or its default value.

getMaxDF()
Gets the value of maxDF or its default value.

New in version 2.4.0.

getMinDF()
Gets the value of minDF or its default value.

New in version 1.6.0.

getMinTF()
Gets the value of minTF or its default value.

New in version 1.6.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

getVocabSize()
Gets the value of vocabSize or its default value.

New in version 1.6.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxDF = *Param(parent='undefined', name='maxDF', doc='Specifies the maximum number of different documents a term could appear in to be included in the vocabulary. A term that appears more than the threshold will be ignored. If this is an integer >= 1, this specifies the maximum number of documents the term could appear in; if this is a double in [0,1), then this specifies the maximum fraction of documents the term could appear in. Default (2^63) - 1')*

minDF = *Param(parent='undefined', name='minDF', doc='Specifies the minimum number of different documents a term must appear in to be included in the vocabulary. If this is an integer >= 1, this specifies the number of documents the term must appear in; if this is a double in [0,1), then this specifies the fraction of documents. Default 1.0')*

minTF = *Param(parent='undefined', name='minTF', doc="Filter to ignore rare words in a document. For each document, terms with frequency/count less than the given threshold are ignored. If this is an integer >= 1, then this specifies a count (of times the term must appear in the document); if this is a double in [0,1), then this specifies a fraction (out of the document's token count). Note that the parameter is only used in transform of CountVectorizerModel and does not affect fitting. Default 1.0")*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses `dir()` to get all attributes of type `Param`.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setBinary(*value*) [source]
Sets the value of `binary`.

New in version 2.0.0.

setInputCol(*value*) [source]
Sets the value of `inputCol`.

setMaxDF(*value*) [source]

Sets the value of **maxDF**.

New in version 2.4.0.

setMinDF(*value*) [source]

Sets the value of **minDF**.

New in version 1.6.0.

setMinTF(*value*) [source]

Sets the value of **minTF**.

New in version 1.6.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *minTF=1.0*, *minDF=1.0*, *maxDF=2 ** 63 - 1*, *vocabSize=1 << 18*, *binary=False*, *inputCol=None*, *outputCol=None*) [source]

Set the params for the CountVectorizer

New in version 1.6.0.

setVocabSize(*value*) [source]

Sets the value of **vocabSize**.

New in version 1.6.0.

vocabSize = *Param(parent='undefined', name='vocabSize', doc='max size of the vocabulary. Default 1 << 18.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**CountVectorizerModel**(*java_model=None*) [source]

Model fitted by **CountVectorizer**.

New in version 1.6.0.

binary = *Param(parent='undefined', name='binary', doc='Binary toggle to control the output vector values. If True, all nonzero counts (after minTF filter applied) are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. Default False')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

classmethod **from_vocabulary**(*vocabulary*, *inputCol*, *outputCol=None*, *minTF=None*, *binary=None*) [source]

Construct the model directly from a vocabulary list of strings, requires an active SparkContext.

New in version 2.4.0.

getBinary()

Gets the value of binary or its default value.

New in version 2.0.0.

getInputCol()

Gets the value of inputCol or its default value.

getMaxDF()

Gets the value of maxDF or its default value.

New in version 2.4.0.

getMinDF()

Gets the value of minDF or its default value.

New in version 1.6.0.

getMinTF()

Gets the value of minTF or its default value.

New in version 1.6.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

getVocabSize()

Gets the value of vocabSize or its default value.

New in version 1.6.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxDF = *Param(parent='undefined', name='maxDF', doc='Specifies the maximum number of different documents a term could appear in to be included in the vocabulary. A term that appears more than the threshold will be ignored. If this is an integer >= 1, this specifies the maximum number of documents the term could appear in; if this is a double in [0,1), then this specifies the maximum fraction of documents the term could appear in. Default (2^63) - 1')*

minDF = *Param(parent='undefined', name='minDF', doc='Specifies the minimum number of different documents a term must appear in to be included in the vocabulary. If this is an integer >= 1, this specifies the number of documents the term must appear in; if this is a double in [0,1), then this specifies the fraction of documents. Default 1.0')*

minTF = *Param(parent='undefined', name='minTF', doc="Filter to ignore rare words in a document. For each document, terms with frequency/count less than the given threshold are ignored. If this is an integer >= 1, then this specifies a count (of times the term must appear in the document); if this is a double in [0,1), then this specifies a fraction (out of the document's token count). Note that the parameter is only used in transform of CountVectorizerModel and does not affect fitting. Default 1.0")*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setBinary(*value*) [source]

Sets the value of **binary**.

New in version 2.4.0.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setMinTF(*value*) [source]

Sets the value of **minTF**.

New in version 2.4.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

vocabSize = *Param(parent='undefined', name='vocabSize', doc='max size of the vocabulary. Default 1 << 18.')*

property **vocabulary**

An array of terms in the vocabulary.

New in version 1.6.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**DCT**(*inverse=False*, *inputCol=None*, *outputCol=None*) [source]

A feature transformer that takes the 1D discrete cosine transform of a real vector. No zero padding is performed on the input vector. It returns a real vector of the same length representing the DCT. The return vector is scaled such that the transform matrix is unitary (aka scaled DCT-II).

See also: More information on Wikipedia.

```
>>> from pyspark.ml.linalg import Vectors
>>> df1 = spark.createDataFrame([(Vectors.dense([5.0, 8.0, 6.0]),)], ["vec"])
>>> dct = DCT( )
>>> dct.setInverse(False)
DCT...
>>> dct.setInputCol("vec")
DCT...
>>> dct.setOutputCol("resultVec")
DCT...
>>> df2 = dct.transform(df1)
>>> df2.head().resultVec
DenseVector([10.969..., -0.707..., -2.041...])
>>> df3 = DCT(inverse=True, inputCol="resultVec", outputCol="origVec").transform(df2)
>>> df3.head().origVec
DenseVector([5.0, 8.0, 6.0])
>>> dctPath = temp_path + "/dct"
>>> dct.save(dctPath)
>>> loadedDtc = DCT.load(dctPath)
>>> loadedDtc.getInverse()
False
```

New in version 1.6.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getInverse() [source]

Gets the value of inverse or its default value.

New in version 1.6.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inverse = *Param(parent='undefined', name='inverse', doc='Set transformer to perform inverse DCT, default False.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setInverse(*value*) [source]

Sets the value of **inverse**.

New in version 1.6.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *inverse=False*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this DCT.

New in version 1.6.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**ElementwiseProduct**(*scalingVec=None*, *inputCol=None*, *outputCol=None*) [source]

Outputs the Hadamard product (i.e., the element-wise product) of each input vector with a provided “weight” vector. In other words, it scales each column of the dataset by a scalar multiplier.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(Vectors.dense([2.0, 1.0, 3.0]),)], ["values"])
>>> ep = ElementwiseProduct()
>>> ep.setScalingVec(Vectors.dense([1.0, 2.0, 3.0]))
ElementwiseProduct...
>>> ep.setInputCol("values")
ElementwiseProduct...
>>> ep.setOutputCol("eprod")
ElementwiseProduct...
>>> ep.transform(df).head().eprod
DenseVector([2.0, 2.0, 9.0])
>>> ep.setParams(scalingVec=Vectors.dense([2.0, 3.0, 5.0])).transform(df).head().eprod
DenseVector([4.0, 3.0, 15.0])
>>> elementwiseProductPath = temp_path + "/elementwise-product"
>>> ep.save(elementwiseProductPath)
>>> loadedEp = ElementwiseProduct.load(elementwiseProductPath)
>>> loadedEp.getScalingVec() == ep.getScalingVec()
True
```

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

getScalingVec() [source]

Gets the value of scalingVec or its default value.

New in version 2.0.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

scalingVec = *Param(parent='undefined', name='scalingVec', doc='Vector for hadamard product.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *scalingVec=None*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this ElementwiseProduct.

New in version 1.5.0.

setScalingVec(*value*) [source]

Sets the value of **scalingVec**.

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**FeatureHasher**(*numFeatures=262144*, *inputCols=None*, *outputCol=None*, *categoricalCols=None*) [source]

Feature hashing projects a set of categorical or numerical features into a feature vector of specified dimension (typically substantially smaller than that of the original feature space). This is done using the hashing trick (://en.wikipedia.org/wiki/Feature_hashing) to map features to indices in the feature vector.

The FeatureHasher transformer operates on multiple columns. Each column may contain either numeric or categorical features. Behavior and handling of column data types is as follows:

- Numeric columns:

  For numeric features, the hash value of the column name is used to map the feature value to its index in the feature vector. By default, numeric features are not treated as categorical (even when they are integers). To treat them as categorical, specify the relevant columns in *categoricalCols*.

- String columns:

  For categorical features, the hash value of the string "column_name=value" is used to map to the vector index, with an indicator value of *1.0*. Thus, categorical features are "one-hot" encoded (similarly to using **OneHotEncoder** with *dropLast=false*).

- Boolean columns:

  Boolean values are treated in the same way as string columns. That is, boolean features are represented as "column_name=true" or "column_name=false", with an indicator value of *1.0*.

Null (missing) values are ignored (implicitly zero in the resulting feature vector).

Since a simple modulo is used to transform the hash function to a vector index, it is advisable to use a power of two as the *numFeatures* parameter; otherwise the features will not be mapped evenly to the vector indices.

```
>>> data = [(2.0, True, "1", "foo"), (3.0, False, "2", "bar")]
>>> cols = ["real", "bool", "stringNum", "string"]
>>> df = spark.createDataFrame(data, cols)
>>> hasher = FeatureHasher()
>>> hasher.setInputCols(cols)
FeatureHasher...
>>> hasher.setOutputCol("features")
FeatureHasher...
>>> hasher.transform(df).head().features
SparseVector(262144, {174475: 2.0, 247670: 1.0, 257907: 1.0, 262126: 1.0})
>>> hasher.setCategoricalCols(["real"]).transform(df).head().features
SparseVector(262144, {171257: 1.0, 247670: 1.0, 257907: 1.0, 262126: 1.0})
>>> hasherPath = temp_path + "/hasher"
>>> hasher.save(hasherPath)
>>> loadedHasher = FeatureHasher.load(hasherPath)
>>> loadedHasher.getNumFeatures() == hasher.getNumFeatures()
True
>>> loadedHasher.transform(df).head().features == hasher.transform(df).head().features
True
```

New in version 2.3.0.

categoricalCols = *Param(parent='undefined', name='categoricalCols', doc='numeric columns to treat as categorical')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy

and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getCategoricalCols() [source]
Gets the value of binary or its default value.

New in version 2.3.0.

getInputCols()
Gets the value of inputCols or its default value.

getNumFeatures()
Gets the value of numFeatures or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numFeatures = *Param(parent='undefined', name='numFeatures', doc='Number of features. Should be greater than 0.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setCategoricalCols(*value*) [source]
Sets the value of **categoricalCols**.

New in version 2.3.0.

setInputCols(*value*) [source]
Sets the value of **inputCols**.

setNumFeatures(*value*) [source]
Sets the value of **numFeatures**.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

setParams(*self*, *numFeatures=1 << 18*, *inputCols=None*, *outputCol=None*, *categoricalCols=None*) [source]
Sets params for this FeatureHasher.

New in version 2.3.0.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**HashingTF**(*numFeatures=262144*, *binary=False*, *inputCol=None*, *outputCol=None*) [source]

Maps a sequence of terms to their term frequencies using the hashing trick. Currently we use Austin Appleby's MurmurHash 3 algorithm (MurmurHash3_x86_32) to calculate the hash code value for the term object. Since a simple modulo is used to transform the hash function to a column index, it is advisable to use a power of two as the numFeatures parameter; otherwise the features will not be mapped evenly to the columns.

```
>>> df = spark.createDataFrame([(["a", "b", "c"],)], ["words"])
>>> hashingTF = HashingTF(inputCol="words", outputCol="features")
>>> hashingTF.setNumFeatures(10)
HashingTF...
>>> hashingTF.transform(df).head().features
SparseVector(10, {5: 1.0, 7: 1.0, 8: 1.0})
>>> hashingTF.setParams(outputCol="freqs").transform(df).head().freqs
SparseVector(10, {5: 1.0, 7: 1.0, 8: 1.0})
>>> params = {hashingTF.numFeatures: 5, hashingTF.outputCol: "vector"}
>>> hashingTF.transform(df, params).head().vector
SparseVector(5, {0: 1.0, 2: 1.0, 3: 1.0})
>>> hashingTFPath = temp_path + "/hashing-tf"
>>> hashingTF.save(hashingTFPath)
>>> loadedHashingTF = HashingTF.load(hashingTFPath)
>>> loadedHashingTF.getNumFeatures() == hashingTF.getNumFeatures()
True
>>> hashingTF.indexOf("b")
5
```

New in version 1.3.0.

binary = *Param(parent='undefined', name='binary', doc='If True, all non zero counts are set to 1. This is useful for discrete probabilistic models that model binary events rather than integer counts. Default False.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getBinary() [source]

Gets the value of binary or its default value.

New in version 2.0.0.

getInputCol()

Gets the value of inputCol or its default value.

getNumFeatures()

Gets the value of numFeatures or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

indexOf(*term*) [source]

Returns the index of the input term.

New in version 3.0.0.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numFeatures = *Param(parent='undefined', name='numFeatures', doc='Number of features. Should be greater than 0.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setBinary(*value*) [source]

Sets the value of **binary**.

New in version 2.0.0.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setNumFeatures(*value*) [source]

Sets the value of **numFeatures**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *numFeatures=1 << 18*, *binary=False*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this HashingTF.

New in version 1.3.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**IDF**(*minDocFreq=0*, *inputCol=None*, *outputCol=None*) [source]

Compute the Inverse Document Frequency (IDF) given a collection of documents.

```
>>> from pyspark.ml.linalg import DenseVector
>>> df = spark.createDataFrame([(DenseVector([1.0, 2.0]),),
...     (DenseVector([0.0, 1.0]),), (DenseVector([3.0, 0.2]),)], ["tf"])
>>> idf = IDF(minDocFreq=3)
>>> idf.setInputCol("tf")
IDF...
>>> idf.setOutputCol("idf")
IDF...
>>> model = idf.fit(df)
>>> model.setOutputCol("idf")
IDFModel...
>>> model.getMinDocFreq()
3
>>> model.idf
DenseVector([0.0, 0.0])
>>> model.docFreq
[0, 3]
>>> model.numDocs == df.count()
True
>>> model.transform(df).head().idf
DenseVector([0.0, 0.0])
>>> idf.setParams(outputCol="freqs").fit(df).transform(df).collect()[1].freqs
DenseVector([0.0, 0.0])
>>> params = {idf.minDocFreq: 1, idf.outputCol: "vector"}
>>> idf.fit(df, params).transform(df).head().vector
DenseVector([0.2877, 0.0])
>>> idfPath = temp_path + "/idf"
>>> idf.save(idfPath)
>>> loadedIdf = IDF.load(idfPath)
>>> loadedIdf.getMinDocFreq() == idf.getMinDocFreq()
True
>>> modelPath = temp_path + "/idf-model"
>>> model.save(modelPath)
>>> loadedModel = IDFModel.load(modelPath)
>>> loadedModel.transform(df).head().idf == model.transform(df).head().idf
True
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance

Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getMinDocFreq()

Gets the value of minDocFreq or its default value.

New in version 1.4.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

minDocFreq = *Param(parent='undefined', name='minDocFreq', doc='minimum number of documents in which a term should appear for filtering')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setMinDocFreq(*value*) [source]

Sets the value of **minDocFreq**.

New in version 1.4.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *minDocFreq=0*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this IDF.

New in version 1.4.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**IDFModel**(*java_model=None*) [source]

Model fitted by **IDF**.

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

property **docFreq**

Returns the document frequency.

New in version 3.0.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getMinDocFreq()

Gets the value of minDocFreq or its default value.

New in version 1.4.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

property **idf**

Returns the IDF vector.

New in version 2.0.0.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

minDocFreq = *Param(parent='undefined', name='minDocFreq', doc='minimum number of documents in which a term should appear for filtering')*

property **numDocs**

Returns number of documents evaluated to compute idf

New in version 3.0.0.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**Imputer**(*strategy='mean'*, *missingValue=nan*, *inputCols=None*, *outputCols=None*, *inputCol=None*, *outputCol=None*, *relativeError=0.001*) [source]

Imputation estimator for completing missing values, either using the mean or the median of the columns in which the missing values are located. The input columns should be of DoubleType or FloatType. Currently Imputer does not support categorical features and possibly creates incorrect values for a categorical feature.

Note that the mean/median value is computed after filtering out missing values. All Null values in the input columns are treated as missing, and so are also imputed. For computing median, **pyspark.sql.DataFrame.approxQuantile()** is used with a relative error of *0.001*.

```
>>> df = spark.createDataFrame([(1.0, float("nan")), (2.0, float("nan")), (float("nan"), 3.0),
...                             (4.0, 4.0), (5.0, 5.0)], ["a", "b"])
>>> imputer = Imputer()
>>> imputer.setInputCols(["a", "b"])
Imputer...
>>> imputer.setOutputCols(["out_a", "out_b"])
Imputer...
>>> imputer.getRelativeError()
0.001
>>> model = imputer.fit(df)
>>> model.setInputCols(["a", "b"])
ImputerModel...
>>> model.getStrategy()
'mean'
>>> model.surrogateDF.show()
+---+---+
|  a|  b|
+---+---+
|3.0|4.0|
+---+---+
...
>>> model.transform(df).show()
+---+---+-----+-----+
|  a|  b|out_a|out_b|
+---+---+-----+-----+
|1.0|NaN|  1.0|  4.0|
|2.0|NaN|  2.0|  4.0|
|NaN|3.0|  3.0|  3.0|
...
>>> imputer.setStrategy("median").setMissingValue(1.0).fit(df).transform(df).show()
+---+---+-----+-----+
|  a|  b|out_a|out_b|
+---+---+-----+-----+
|1.0|NaN|  4.0|  NaN|
...
>>> df1 = spark.createDataFrame([(1.0,), (2.0,), (float("nan"),), (4.0,), (5.0,)], ["a"])
>>> imputer1 = Imputer(inputCol="a", outputCol="out_a")
>>> model1 = imputer1.fit(df1)
>>> model1.surrogateDF.show()
+---+
|  a|
+---+
|3.0|
+---+
...
>>> model1.transform(df1).show()
+---+-----+
|  a|out_a|
+---+-----+
|1.0|  1.0|
|2.0|  2.0|
|NaN|  3.0|
...
>>> imputer1.setStrategy("median").setMissingValue(1.0).fit(df1).transform(df1).show()
+---+-----+
|  a|out_a|
+---+-----+
|1.0|  4.0|
...
>>> df2 = spark.createDataFrame([(float("nan"),), (float("nan"),), (3.0,), (4.0,), (5.0,)],
...                             ["b"])
>>> imputer2 = Imputer(inputCol="b", outputCol="out_b")
>>> model2 = imputer2.fit(df2)
>>> model2.surrogateDF.show()
+---+
|  b|
+---+
|4.0|
+---+
...
>>> model2.transform(df2).show()
+---+-----+
|  b|out_b|
+---+-----+
|NaN|  4.0|
|NaN|  4.0|
|3.0|  3.0|
...
>>> imputer2.setStrategy("median").setMissingValue(1.0).fit(df2).transform(df2).show()
+---+-----+
|  b|out_b|
+---+-----+
|NaN|  NaN|
...
>>> imputerPath = temp_path + "/imputer"
>>> imputer.save(imputerPath)
>>> loadedImputer = Imputer.load(imputerPath)
>>> loadedImputer.getStrategy() == imputer.getStrategy()
True
>>> loadedImputer.getMissingValue()
1.0
>>> modelPath = temp_path + "/imputer-model"
>>> model.save(modelPath)
>>> loadedModel = ImputerModel.load(modelPath)
>>> loadedModel.transform(df).head().out_a == model.transform(df).head().out_a
True
```

New in version 2.2.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getInputCols()

Gets the value of inputCols or its default value.

getMissingValue()

Gets the value of **missingValue** or its default value.

New in version 2.2.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getOutputCols()

Gets the value of outputCols or its default value.

getParam(*paramName*)

Gets a param by its name.

getRelativeError()

Gets the value of relativeError or its default value.

getStrategy()

Gets the value of **strategy** or its default value.

New in version 2.2.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

missingValue = *Param(parent='undefined', name='missingValue', doc='The placeholder for the missing values. All occurrences of missingValue will be imputed.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

relativeError = *Param(parent='undefined', name='relativeError', doc='the relative target precision for the approximate quantile algorithm. Must be in the range [0, 1]')*

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setInputCols(*value*) [source]

Sets the value of **inputCols**.

New in version 2.2.0.

setMissingValue(*value*) [source]

Sets the value of **missingValue**.

New in version 2.2.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

setOutputCols(*value*) [source]

Sets the value of **outputCols**.

New in version 2.2.0.

setParams(*self*, *strategy="mean"*, *missingValue=float("nan")*, *inputCols=None*, *outputCols=None*, *inputCol=None*, *outputCol=None*, *relativeError=0.001*) [source]

Sets params for this Imputer.

New in version 2.2.0.

setRelativeError(*value*) [source]

Sets the value of **relativeError**.

New in version 3.0.0.

setStrategy(*value*) [source]

Sets the value of **strategy**.

New in version 2.2.0.

strategy = *Param(parent='undefined', name='strategy', doc='strategy for imputation. If mean, then replace missing values using the mean value of the feature. If median, then replace missing values using the median value of the feature.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**ImputerModel**(*java_model=None*) [source]

Model fitted by **Imputer**.

New in version 2.2.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:

merged param map

getInputCol()
Gets the value of inputCol or its default value.

getInputCols()
Gets the value of inputCols or its default value.

getMissingValue()
Gets the value of **missingValue** or its default value.

New in version 2.2.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getOutputCols()
Gets the value of outputCols or its default value.

getParam(*paramName*)
Gets a param by its name.

getRelativeError()
Gets the value of relativeError or its default value.

getStrategy()
Gets the value of **strategy** or its default value.

New in version 2.2.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

missingValue = *Param(parent='undefined', name='missingValue', doc='The placeholder for the missing values. All occurrences of missingValue will be imputed.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

relativeError = *Param(parent='undefined', name='relativeError', doc='the relative target precision for the approximate quantile algorithm. Must be in the range [0, 1]')*

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCols(*value*) [source]
Sets the value of **inputCols**.

New in version 3.0.0.

setOutputCols(*value*) [source]
Sets the value of **outputCols**.

New in version 3.0.0.

strategy = *Param(parent='undefined', name='strategy', doc='strategy for imputation. If mean, then replace missing values using the mean value of the feature. If median, then replace missing values using the median value of the feature.')*

property **surrogateDF**
Returns a DataFrame containing inputCols and their corresponding surrogates, which are used to replace the missing values in the input DataFrame.

New in version 2.2.0.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**IndexToString**(*inputCol=None*, *outputCol=None*, *labels=None*) [source]

A **Transformer** that maps a column of indices back to a new column of corresponding string values. The index-string mapping is either from the ML attributes of the input column, or from user-supplied labels (which take precedence over ML attributes). See **StringIndexer** for converting strings into indices.

New in version 1.6.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getLabels() [source]

Gets the value of **labels** or its default value.

New in version 1.6.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labels = *Param(parent='undefined', name='labels', doc='Optional array of labels specifying index-string mapping. If not provided or if empty, then metadata from inputCol is used instead.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setLabels(*value*) [source]

Sets the value of **labels**.

New in version 1.6.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *inputCol=None*, *outputCol=None*, *labels=None*) [source]

Sets params for this IndexToString.

New in version 1.6.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**Interaction**(*inputCols=None*, *outputCol=None*) [source]

Implements the feature interaction transform. This transformer takes in Double and Vector type columns and outputs a flattened vector of their feature interactions. To handle interaction, we first one-hot encode any nominal features. Then, a vector of the feature cross-products is produced.

For example, given the input feature values *Double(2)* and *Vector(3, 4)*, the output would be *Vector(6, 8)* if all input features were numeric. If the first feature was instead nominal with four categories, the output would then be *Vector(0, 0, 0, 0, 3, 4, 0, 0)*.

```
>>> df = spark.createDataFrame([(0.0, 1.0), (2.0, 3.0)], ["a", "b"])
>>> interaction = Interaction()
>>> interaction.setInputCols(["a", "b"])
Interaction...
>>> interaction.setOutputCol("ab")
Interaction...
>>> interaction.transform(df).show()
+---+---+-----+
|  a|  b|   ab|
+---+---+-----+
|0.0|1.0|[0.0]|
|2.0|3.0|[6.0]|
+---+---+-----+
...
>>> interactionPath = temp_path + "/interaction"
>>> interaction.save(interactionPath)
>>> loadedInteraction = Interaction.load(interactionPath)
>>> loadedInteraction.transform(df).head().ab == interaction.transform(df).head().ab
True
```

New in version 3.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCols()

Gets the value of inputCols or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCols(*value*) [source]

Sets the value of **inputCols**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

setParams(*self*, *inputCols=None*, *outputCol=None*) [source]

Sets params for this Interaction.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**MaxAbsScaler**(*inputCol=None*, *outputCol=None*) [source]

Rescale each feature individually to range [-1, 1] by dividing through the largest maximum absolute value in each feature. It does not shift/center the data, and thus does not destroy any sparsity.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(Vectors.dense([1.0]),), (Vectors.dense([2.0]),)], ["a"])
>>> maScaler = MaxAbsScaler(outputCol="scaled")
>>> maScaler.setInputCol("a")
MaxAbsScaler...
>>> model = maScaler.fit(df)
>>> model.setOutputCol("scaledOutput")
MaxAbsScalerModel...
>>> model.transform(df).show()
+-----+------------+
|    a|scaledOutput|
+-----+------------+
|[1.0]|       [0.5]|
|[2.0]|       [1.0]|
+-----+------------+
...
>>> scalerPath = temp_path + "/max-abs-scaler"
>>> maScaler.save(scalerPath)
>>> loadedMAScaler = MaxAbsScaler.load(scalerPath)
>>> loadedMAScaler.getInputCol() == maScaler.getInputCol()
True
>>> loadedMAScaler.getOutputCol() == maScaler.getOutputCol()
True
>>> modelPath = temp_path + "/max-abs-scaler-model"
>>> model.save(modelPath)
>>> loadedModel = MaxAbsScalerModel.load(modelPath)
>>> loadedModel.maxAbs == model.maxAbs
True
```

New in version 2.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**

- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)
Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getInputCol()
Gets the value of inputCol or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

setParams(*self*, *inputCol=None*, *outputCol=None*) [source]
Sets params for this MaxAbsScaler.

New in version 2.0.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**MaxAbsScalerModel**(*java_model=None*) [source]
Model fitted by **MaxAbsScaler**.

New in version 2.0.0.

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()
Gets the value of inputCol or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **maxAbs**
Max Abs vector.

New in version 2.0.0.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

New in version 3.0.0.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**MinHashLSH**(*inputCol=None*, *outputCol=None*, *seed=None*, *numHashTables=1*) [source]
LSH class for Jaccard distance. The input can be dense or sparse vectors, but it is more efficient if it is sparse. For example, *Vectors.sparse(10, [(2, 1.0), (3, 1.0), (5, 1.0)])* means there are 10 elements in the space. This set contains elements 2, 3, and 5. Also, any input vector must have at least 1 non-zero index, and all non-zero values are treated as binary "1" values.

See also: Wikipedia on MinHash

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.sql.functions import col
>>> data = [(0, Vectors.sparse(6, [0, 1, 2], [1.0, 1.0, 1.0]),),
...         (1, Vectors.sparse(6, [2, 3, 4], [1.0, 1.0, 1.0]),),
...         (2, Vectors.sparse(6, [0, 2, 4], [1.0, 1.0, 1.0]),)]
>>> df = spark.createDataFrame(data, ["id", "features"])
>>> mh = MinHashLSH()
>>> mh.setInputCol("features")
MinHashLSH...
>>> mh.setOutputCol("hashes")
MinHashLSH...
>>> mh.setSeed(12345)
MinHashLSH...
>>> model = mh.fit(df)
>>> model.setInputCol("features")
MinHashLSHModel...
>>> model.transform(df).head()
Row(id=0, features=SparseVector(6, {0: 1.0, 1: 1.0, 2: 1.0}), hashes=[DenseVector([6179668...
>>> data2 = [(3, Vectors.sparse(6, [1, 3, 5], [1.0, 1.0, 1.0]),),
...          (4, Vectors.sparse(6, [2, 3, 5], [1.0, 1.0, 1.0]),),
...          (5, Vectors.sparse(6, [1, 2, 4], [1.0, 1.0, 1.0]),)]
>>> df2 = spark.createDataFrame(data2, ["id", "features"])
>>> key = Vectors.sparse(6, [1, 2], [1.0, 1.0])
>>> model.approxNearestNeighbors(df2, key, 1).collect()
[Row(id=5, features=SparseVector(6, {1: 1.0, 2: 1.0, 4: 1.0}), hashes=[DenseVector([6179668...
>>> model.approxSimilarityJoin(df, df2, 0.6, distCol="JaccardDistance").select(
...     col("datasetA.id").alias("idA"),
...     col("datasetB.id").alias("idB"),
...     col("JaccardDistance")).show()
+---+---+---------------+
|idA|idB|JaccardDistance|
+---+---+---------------+
|  0|  5|            0.5|
|  1|  4|            0.5|
+---+---+---------------+
...
>>> mhPath = temp_path + "/mh"
>>> mh.save(mhPath)
>>> mh2 = MinHashLSH.load(mhPath)
>>> mh2.getOutputCol() == mh.getOutputCol()
True
>>> modelPath = temp_path + "/mh-model"
>>> model.save(modelPath)
>>> model2 = MinHashLSHModel.load(modelPath)
>>> model.transform(df).head().hashes == model2.transform(df).head().hashes
True
```

New in version 2.2.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getNumHashTables()

Gets the value of numHashTables or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

getSeed()
Gets the value of seed or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numHashTables = *Param(parent='undefined', name='numHashTables', doc='number of hash tables, where increasing number of hash tables lowers the false negative rate, and decreasing it improves the running performance.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*)
Sets the value of **inputCol**.

setNumHashTables(*value*)
Sets the value of **numHashTables**.

setOutputCol(*value*)
Sets the value of **outputCol**.

setParams(*self*, *inputCol=None*, *outputCol=None*, *seed=None*, *numHashTables=1*) [source]
Sets params for this MinHashLSH.

New in version 2.2.0.

setSeed(*value*) [source]
Sets the value of **seed**.

write()
Returns an MLWriter instance for this ML instance.

class `pyspark.ml.feature.`**MinHashLSHModel**(*java_model=None*) [source]
Model produced by **MinHashLSH**, where where multiple hash functions are stored. Each hash function is picked from the following family of hash functions, where \(a_i\) and \(b_i\) are randomly chosen integers less than prime: \(h_i(x) = ((x \cdot a_i + b_i) \mod prime)\) This hash family is approximately min-wise independent according to the reference.

See also: Tom Bohman, Colin Cooper, and Alan Frieze. "Min-wise independent linear permutations." Electronic Journal of Combinatorics 7 (2000): R26.

New in version 2.2.0.

approxNearestNeighbors(*dataset*, *key*, *numNearestNeighbors*, *distCol='distCol'*)
Given a large dataset and an item, approximately find at most k items which have the closest distance to the item. If the **outputCol** is missing, the method will transform the data; if the **outputCol** exists, it will use that. This allows caching of the transformed data when necessary.

Note: This method is experimental and will likely change behavior in the next release.

Parameters:
- **dataset** – The dataset to search for nearest neighbors of the key.
- **key** – Feature vector representing the item to search for.
- **numNearestNeighbors** – The maximum number of nearest neighbors.
- **distCol** – Output column for storing the distance between each result row and the key. Use "distCol" as default value if it's not specified.

Returns:
A dataset containing at most k items closest to the key. A column "distCol" is added to show the distance between each row and the key.

approxSimilarityJoin(*datasetA*, *datasetB*, *threshold*, *distCol='distCol'*)
Join two datasets to approximately find all pairs of rows whose distance are smaller than the threshold. If the **outputCol** is missing, the method will transform the data; if the **outputCol** exists, it will use that. This allows caching of the transformed data when necessary.

Parameters:
- **datasetA** – One of the datasets to join.
- **datasetB** – Another dataset to join.
- **threshold** – The threshold for the distance of row pairs.
- **distCol** – Output column for storing the distance between each pair of rows. Use "distCol" as default value if it's not specified.

Returns:
A joined dataset containing pairs of rows. The original rows are in columns “datasetA” and “datasetB”, and a column “distCol” is added to show the distance between each pair.

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()
Gets the value of inputCol or its default value.

getNumHashTables()
Gets the value of numHashTables or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numHashTables = *Param(parent='undefined', name='numHashTables', doc='number of hash tables, where increasing number of hash tables lowers the false negative rate, and decreasing it improves the running performance.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of ‘write().save(path)’.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*)
Sets the value of **inputCol**.

setOutputCol(*value*)
Sets the value of **outputCol**.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**MinMaxScaler**(*min=0.0*, *max=1.0*, *inputCol=None*, *outputCol=None*) [source]

Rescale each feature individually to a common range [min, max] linearly using column summary statistics, which is also known as min-max normalization or Rescaling. The rescaled value for feature E is calculated as,

Rescaled(e_i) = (e_i - E_min) / (E_max - E_min) * (max - min) + min

For the case E_max == E_min, Rescaled(e_i) = 0.5 * (max + min)

Note: Since zero values will probably be transformed to non-zero values, output of the transformer will be DenseVector even for sparse input.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(Vectors.dense([0.0]),), (Vectors.dense([2.0]),)], ["a"])
>>> mmScaler = MinMaxScaler(outputCol="scaled")
>>> mmScaler.setInputCol("a")
MinMaxScaler...
>>> model = mmScaler.fit(df)
>>> model.setOutputCol("scaledOutput")
MinMaxScalerModel...
>>> model.originalMin
DenseVector([0.0])
>>> model.originalMax
DenseVector([2.0])
>>> model.transform(df).show()
+-----+------------+
|    a|scaledOutput|
+-----+------------+
|[0.0]|       [0.0]|
|[2.0]|       [1.0]|
+-----+------------+
...
>>> minMaxScalerPath = temp_path + "/min-max-scaler"
>>> mmScaler.save(minMaxScalerPath)
>>> loadedMMScaler = MinMaxScaler.load(minMaxScalerPath)
>>> loadedMMScaler.getMin() == mmScaler.getMin()
True
>>> loadedMMScaler.getMax() == mmScaler.getMax()
True
>>> modelPath = temp_path + "/min-max-scaler-model"
>>> model.save(modelPath)
>>> loadedModel = MinMaxScalerModel.load(modelPath)
>>> loadedModel.originalMin == model.originalMin
True
>>> loadedModel.originalMax == model.originalMax
True
```

New in version 1.6.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getMax()

Gets the value of max or its default value.

New in version 1.6.0.

getMin()
Gets the value of min or its default value.

New in version 1.6.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

max = *Param(parent='undefined', name='max', doc='Upper bound of the output feature range')*

min = *Param(parent='undefined', name='min', doc='Lower bound of the output feature range')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setMax(*value*) [source]
Sets the value of **max**.

New in version 1.6.0.

setMin(*value*) [source]
Sets the value of **min**.

New in version 1.6.0.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

setParams(*self*, *min=0.0*, *max=1.0*, *inputCol=None*, *outputCol=None*) [source]
Sets params for this MinMaxScaler.

New in version 1.6.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**MinMaxScalerModel**(*java_model=None*) [source]
Model fitted by **MinMaxScaler**.

New in version 1.6.0.

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values

Returns:
merged param map

getInputCol()
Gets the value of inputCol or its default value.

getMax()
Gets the value of max or its default value.

New in version 1.6.0.

getMin()
Gets the value of min or its default value.

New in version 1.6.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

max = *Param(parent='undefined', name='max', doc='Upper bound of the output feature range')*

min = *Param(parent='undefined', name='min', doc='Lower bound of the output feature range')*

property **originalMax**
Max value for each original column during fitting.

New in version 2.0.0.

property **originalMin**
Min value for each original column during fitting.

New in version 2.0.0.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

New in version 3.0.0.

setMax(*value*) [source]
Sets the value of **max**.

New in version 3.0.0.

setMin(*value*) [source]
Sets the value of **min**.

New in version 3.0.0.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**NGram**(*n=2*, *inputCol=None*, *outputCol=None*) [source]

A feature transformer that converts the input array of strings into an array of n-grams. Null values in the input array are ignored. It returns an array of n-grams where each n-gram is represented by a space-separated string of words. When the input is empty, an empty array is returned. When the input array length is less than n (number of elements per n-gram), no n-grams are returned.

```
>>> df = spark.createDataFrame([Row(inputTokens=["a", "b", "c", "d", "e"])])
>>> ngram = NGram(n=2)
>>> ngram.setInputCol("inputTokens")
NGram...
>>> ngram.setOutputCol("nGrams")
NGram...
>>> ngram.transform(df).head()
Row(inputTokens=['a', 'b', 'c', 'd', 'e'], nGrams=['a b', 'b c', 'c d', 'd e'])
>>> # Change n-gram length
>>> ngram.setParams(n=4).transform(df).head()
Row(inputTokens=['a', 'b', 'c', 'd', 'e'], nGrams=['a b c d', 'b c d e'])
>>> # Temporarily modify output column.
>>> ngram.transform(df, {ngram.outputCol: "output"}).head()
Row(inputTokens=['a', 'b', 'c', 'd', 'e'], output=['a b c d', 'b c d e'])
>>> ngram.transform(df).head()
Row(inputTokens=['a', 'b', 'c', 'd', 'e'], nGrams=['a b c d', 'b c d e'])
>>> # Must use keyword arguments to specify params.
>>> ngram.setParams("text")
Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
>>> ngramPath = temp_path + "/ngram"
>>> ngram.save(ngramPath)
>>> loadedNGram = NGram.load(ngramPath)
>>> loadedNGram.getN() == ngram.getN()
True
```

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getN() [source]

Gets the value of n or its default value.

New in version 1.5.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

n = *Param(parent='undefined', name='n', doc='number of elements per n-gram (>=1)')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setN(*value*) [source]

Sets the value of **n**.

New in version 1.5.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *n=2*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this NGram.

New in version 1.5.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**Normalizer**(*p=2.0*, *inputCol=None*, *outputCol=None*) [source]

Normalize a vector to have unit norm using the given p-norm.

```
>>> from pyspark.ml.linalg import Vectors
>>> svec = Vectors.sparse(4, {1: 4.0, 3: 3.0})
>>> df = spark.createDataFrame([(Vectors.dense([3.0, -4.0]), svec)], ["dense", "sparse"])
>>> normalizer = Normalizer(p=2.0)
>>> normalizer.setInputCol("dense")
Normalizer...
>>> normalizer.setOutputCol("features")
Normalizer...
>>> normalizer.transform(df).head().features
DenseVector([0.6, -0.8])
>>> normalizer.setParams(inputCol="sparse", outputCol="freqs").transform(df).head().freqs
SparseVector(4, {1: 0.8, 3: 0.6})
>>> params = {normalizer.p: 1.0, normalizer.inputCol: "dense", normalizer.outputCol: "vector"}
>>> normalizer.transform(df, params).head().vector
DenseVector([0.4286, -0.5714])
>>> normalizerPath = temp_path + "/normalizer"
>>> normalizer.save(normalizerPath)
>>> loadedNormalizer = Normalizer.load(normalizerPath)
>>> loadedNormalizer.getP() == normalizer.getP()
True
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getP() [source]

Gets the value of p or its default value.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

p = *Param(parent='undefined', name='p', doc='the p norm value.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setP(*value*) [source]

Sets the value of **p**.

New in version 1.4.0.

setParams(*self*, *p=2.0*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this Normalizer.

New in version 1.4.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**OneHotEncoder**(*inputCols=None*, *outputCols=None*, *handleInvalid='error'*, *dropLast=True*, *inputCol=None*, *outputCol=None*) [source]

A one-hot encoder that maps a column of category indices to a column of binary vectors, with at most a single one-value per row that indicates the input category index. For example with 5 categories, an input value of 2.0 would map to an output vector of *[0.0, 0.0, 1.0, 0.0]*. The last category is not included by default (configurable via **dropLast**), because it makes the vector entries sum up to one, and hence linearly dependent. So an input value of 4.0 maps to *[0.0, 0.0, 0.0, 0.0]*.

Note: This is different from scikit-learn's OneHotEncoder, which keeps all categories. The output vectors are sparse.

When **handleInvalid** is configured to 'keep', an extra "category" indicating invalid values is added as last category. So when **dropLast** is true, invalid values are encoded as all-zeros vector.

Note: When encoding multi-column by using **inputCols** and **outputCols** params, input/output cols come in pairs, specified by the order in the arrays, and each pair is treated independently.

See also: **StringIndexer** for converting categorical values into category indices

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(0.0,), (1.0,), (2.0,)], ["input"])
>>> ohe = OneHotEncoder()
>>> ohe.setInputCols(["input"])
OneHotEncoder...
>>> ohe.setOutputCols(["output"])
OneHotEncoder...
>>> model = ohe.fit(df)
>>> model.setOutputCols(["output"])
OneHotEncoderModel...
>>> model.getHandleInvalid()
'error'
>>> model.transform(df).head().output
SparseVector(2, {0: 1.0})
>>> single_col_ohe = OneHotEncoder(inputCol="input", outputCol="output")
>>> single_col_model = single_col_ohe.fit(df)
>>> single_col_model.transform(df).head().output
SparseVector(2, {0: 1.0})
>>> ohePath = temp_path + "/ohe"
>>> ohe.save(ohePath)
>>> loadedOHE = OneHotEncoder.load(ohePath)
>>> loadedOHE.getInputCols() == ohe.getInputCols()
True
>>> modelPath = temp_path + "/ohe-model"
>>> model.save(modelPath)
>>> loadedModel = OneHotEncoderModel.load(modelPath)
>>> loadedModel.categorySizes == model.categorySizes
True
```

New in version 2.3.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

dropLast = *Param(parent='undefined', name='dropLast', doc='whether to drop the last category')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getDropLast()

Gets the value of dropLast or its default value.

New in version 2.3.0.

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getInputCol()

Gets the value of inputCol or its default value.

getInputCols()

Gets the value of inputCols or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getOutputCols()

Gets the value of outputCols or its default value.

getParam(*paramName*)

Gets a param by its name.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="How to handle invalid data during transform(). Options are 'keep' (invalid data presented as an extra categorical feature) or error (throw an error). Note that this Param is only used during transform; during fitting, invalid data will result in an error.")*

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setDropLast(*value*) [source]

Sets the value of **dropLast**.

New in version 2.3.0.

setHandleInvalid(*value*) [source]

Sets the value of **handleInvalid**.

New in version 3.0.0.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setInputCols(*value*) [source]

Sets the value of **inputCols**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

setOutputCols(*value*) [source]

Sets the value of **outputCols**.

New in version 3.0.0.

setParams(*self*, *inputCols=None*, *outputCols=None*, *handleInvalid="error"*, *dropLast=True*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this OneHotEncoder.

New in version 2.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**OneHotEncoderModel**(*java_model=None*) [source]

Model fitted by **OneHotEncoder**.

New in version 2.3.0.

property **categorySizes**

Original number of categories for each feature being encoded. The array contains one value for each input column, in order.

New in version 2.3.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

dropLast = *Param(parent='undefined', name='dropLast', doc='whether to drop the last category')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getDropLast()

Gets the value of dropLast or its default value.

New in version 2.3.0.

getHandleInvalid()
Gets the value of handleInvalid or its default value.

getInputCol()
Gets the value of inputCol or its default value.

getInputCols()
Gets the value of inputCols or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getOutputCols()
Gets the value of outputCols or its default value.

getParam(*paramName*)
Gets a param by its name.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="How to handle invalid data during transform(). Options are 'keep' (invalid data presented as an extra categorical feature) or error (throw an error). Note that this Param is only used during transform; during fitting, invalid data will result in an error.")*

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setDropLast(*value*) [source]
Sets the value of **dropLast**.

New in version 3.0.0.

setHandleInvalid(*value*) [source]
Sets the value of **handleInvalid**.

New in version 3.0.0.

setInputCols(*value*) [source]
Sets the value of **inputCols**.

New in version 3.0.0.

setOutputCols(*value*) [source]
Sets the value of **outputCols**.

New in version 3.0.0.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**PCA**(*k=None*, *inputCol=None*, *outputCol=None*) [source]
PCA trains a model to project vectors to a lower dimensional space of the top **k** principal components.

```
>>> from pyspark.ml.linalg import Vectors
>>> data = [(Vectors.sparse(5, [(1, 1.0), (3, 7.0)]),),
...     (Vectors.dense([2.0, 0.0, 3.0, 4.0, 5.0]),),
...     (Vectors.dense([4.0, 0.0, 0.0, 6.0, 7.0]),)]
>>> df = spark.createDataFrame(data,["features"])
>>> pca = PCA(k=2, inputCol="features")
>>> pca.setOutputCol("pca_features")
PCA...
>>> model = pca.fit(df)
>>> model.getK()
2
>>> model.setOutputCol("output")
PCAModel...
>>> model.transform(df).collect()[0].output
DenseVector([1.648..., -4.013...])
>>> model.explainedVariance
DenseVector([0.794..., 0.205...])
>>> pcaPath = temp_path + "/pca"
>>> pca.save(pcaPath)
>>> loadedPca = PCA.load(pcaPath)
>>> loadedPca.getK() == pca.getK()
True
>>> modelPath = temp_path + "/pca-model"
>>> model.save(modelPath)
>>> loadedModel = PCAModel.load(modelPath)
>>> loadedModel.pc == model.pc
True
>>> loadedModel.explainedVariance == model.explainedVariance
True
```

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getK()

Gets the value of k or its default value.

New in version 1.5.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='the number of principal components')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setK(*value*) [source]
Sets the value of **k**.

New in version 1.5.0.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

setParams(*self*, *k=None*, *inputCol=None*, *outputCol=None*) [source]
Set params for this PCA.

New in version 1.5.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**PCAModel**(*java_model=None*) [source]
Model fitted by **PCA**. Transforms vectors to a lower dimensional space.

New in version 1.5.0.

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

property **explainedVariance**
Returns a vector of proportions of variance explained by each principal component.

New in version 2.0.0.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()
Gets the value of inputCol or its default value.

getK()
Gets the value of k or its default value.

New in version 1.5.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='the number of principal components')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

property **pc**

Returns a principal components Matrix. Each column is one principal component.

New in version 2.0.0.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**PolynomialExpansion**(*degree=2*, *inputCol=None*, *outputCol=None*) [source]

Perform feature expansion in a polynomial space. As said in wikipedia of Polynomial Expansion, "In mathematics, an expansion of a product of sums expresses it as a sum of products by using the fact that multiplication distributes over addition". Take a 2-variable feature vector as an example: *(x, y)*, if we want to expand it with degree 2, then we get *(x, x * x, y, x * y, y * y)*.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(Vectors.dense([0.5, 2.0]),)], ["dense"])
>>> px = PolynomialExpansion(degree=2)
>>> px.setInputCol("dense")
PolynomialExpansion...
>>> px.setOutputCol("expanded")
PolynomialExpansion...
>>> px.transform(df).head().expanded
DenseVector([0.5, 0.25, 2.0, 1.0, 4.0])
>>> px.setParams(outputCol="test").transform(df).head().test
DenseVector([0.5, 0.25, 2.0, 1.0, 4.0])
>>> polyExpansionPath = temp_path + "/poly-expansion"
>>> px.save(polyExpansionPath)
>>> loadedPx = PolynomialExpansion.load(polyExpansionPath)
>>> loadedPx.getDegree() == px.getDegree()
True
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

degree = *Param(parent='undefined', name='degree', doc='the polynomial degree to expand (>= 1)')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getDegree() [source]

Gets the value of degree or its default value.

New in version 1.4.0.

getInputCol()

Gets the value of inputCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setDegree(*value*) [source]

Sets the value of **degree**.

New in version 1.4.0.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *degree=2*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this PolynomialExpansion.

New in version 1.4.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**QuantileDiscretizer**(*numBuckets=2*, *inputCol=None*, *outputCol=None*, *relativeError=0.001*, *handleInvalid='error'*, *numBucketsArray=None*, *inputCols=None*, *outputCols=None*) [source]

QuantileDiscretizer takes a column with continuous features and outputs a column with binned categorical features. The number of bins can be set using the **numBuckets** parameter. It is possible that the number of buckets used will be less than this value, for example, if there are too few distinct values of the input to create enough distinct quantiles. Since 3.0.0, **QuantileDiscretizer** can map multiple columns at once by setting the **inputCols** parameter. If both of the **inputCol** and **inputCols** parameters are set, an Exception will be thrown. To specify the number of buckets for each column, the **numBucketsArray** parameter can be set, or if the number of buckets should be the same across columns, **numBuckets** can be set as a convenience.

NaN handling: Note also that **QuantileDiscretizer** will raise an error when it finds NaN values in the dataset, but the user can also choose to either keep or remove NaN values within the dataset by setting **handleInvalid** parameter. If the user chooses to keep NaN values, they will be handled specially and placed into their own bucket, for example, if 4 buckets are used, then non-NaN data will be put into buckets[0-3], but NaNs will be counted in a special bucket[4].

Algorithm: The bin ranges are chosen using an approximate algorithm (see the documentation for **`approxQuantile()`** for a detailed description). The precision of the approximation can be controlled with the **`relativeError`** parameter. The lower and upper bin bounds will be *-Infinity* and *+Infinity*, covering all real values.

```
>>> values = [(0.1,), (0.4,), (1.2,), (1.5,), (float("nan"),), (float("nan"),)]
>>> df1 = spark.createDataFrame(values, ["values"])
>>> qds1 = QuantileDiscretizer(inputCol="values", outputCol="buckets")
>>> qds1.setNumBuckets(2)
QuantileDiscretizer...
>>> qds1.setRelativeError(0.01)
QuantileDiscretizer...
>>> qds1.setHandleInvalid("error")
QuantileDiscretizer...
>>> qds1.getRelativeError()
0.01
>>> bucketizer = qds1.fit(df1)
>>> qds1.setHandleInvalid("keep").fit(df1).transform(df1).count()
6
>>> qds1.setHandleInvalid("skip").fit(df1).transform(df1).count()
4
>>> splits = bucketizer.getSplits()
>>> splits[0]
-inf
>>> print("%2.1f" % round(splits[1], 1))
0.4
>>> bucketed = bucketizer.transform(df1).head()
>>> bucketed.buckets
0.0
>>> quantileDiscretizerPath = temp_path + "/quantile-discretizer"
>>> qds1.save(quantileDiscretizerPath)
>>> loadedQds = QuantileDiscretizer.load(quantileDiscretizerPath)
>>> loadedQds.getNumBuckets() == qds1.getNumBuckets()
True
>>> inputs = [(0.1, 0.0), (0.4, 1.0), (1.2, 1.3), (1.5, 1.5),
...     (float("nan"), float("nan")), (float("nan"), float("nan"))]
>>> df2 = spark.createDataFrame(inputs, ["input1", "input2"])
>>> qds2 = QuantileDiscretizer(relativeError=0.01, handleInvalid="error", numBuckets=2,
...     inputCols=["input1", "input2"], outputCols=["output1", "output2"])
>>> qds2.getRelativeError()
0.01
>>> qds2.setHandleInvalid("keep").fit(df2).transform(df2).show()
+------+------+-------+-------+
|input1|input2|output1|output2|
+------+------+-------+-------+
|   0.1|   0.0|    0.0|    0.0|
|   0.4|   1.0|    1.0|    1.0|
|   1.2|   1.3|    1.0|    1.0|
|   1.5|   1.5|    1.0|    1.0|
|   NaN|   NaN|    2.0|    2.0|
|   NaN|   NaN|    2.0|    2.0|
+------+------+-------+-------+
...
>>> qds3 = QuantileDiscretizer(relativeError=0.01, handleInvalid="error",
...     numBucketsArray=[5, 10], inputCols=["input1", "input2"],
...     outputCols=["output1", "output2"])
>>> qds3.setHandleInvalid("skip").fit(df2).transform(df2).show()
+------+------+-------+-------+
|input1|input2|output1|output2|
+------+------+-------+-------+
|   0.1|   0.0|    1.0|    1.0|
|   0.4|   1.0|    2.0|    2.0|
|   1.2|   1.3|    3.0|    3.0|
|   1.5|   1.5|    4.0|    4.0|
+------+------+-------+-------+
...
```

New in version 2.0.0.

`clear`(*param*)

Clears a param from the param map if it has been explicitly set.

`copy`(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

`explainParam`(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

`explainParams`()

Returns the documentation of all params with their optionally default values and user-supplied values.

`extractParamMap`(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

`fit`(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **`pyspark.sql.DataFrame`**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

`fitMultiple`(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **`pyspark.sql.DataFrame`**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getHandleInvalid()
Gets the value of handleInvalid or its default value.

getInputCol()
Gets the value of inputCol or its default value.

getInputCols()
Gets the value of inputCols or its default value.

getNumBuckets() [source]
Gets the value of numBuckets or its default value.

New in version 2.0.0.

getNumBucketsArray() [source]
Gets the value of numBucketsArray or its default value.

New in version 3.0.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getOutputCols()
Gets the value of outputCols or its default value.

getParam(*paramName*)
Gets a param by its name.

getRelativeError()
Gets the value of relativeError or its default value.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="how to handle invalid entries. Options are skip (filter out rows with invalid values), error (throw an error), or keep (keep invalid values in a special additional bucket). Note that in the multiple columns case, the invalid handling is applied to all columns. That said for 'error' it will throw an error if any invalids are found in any columns, for 'skip' it will skip rows with any invalids in any columns, etc.")*

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numBuckets = *Param(parent='undefined', name='numBuckets', doc='Maximum number of buckets (quantiles, or categories) into which data points are grouped. Must be >= 2.')*

numBucketsArray = *Param(parent='undefined', name='numBucketsArray', doc='Array of number of buckets (quantiles, or categories) into which data points are grouped. This is for multiple columns input. If transforming multiple columns and numBucketsArray is not set, but numBuckets is set, then numBuckets will be applied across all columns.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

relativeError = *Param(parent='undefined', name='relativeError', doc='the relative target precision for the approximate quantile algorithm. Must be in the range [0, 1]')*

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setHandleInvalid(*value*) [source]
Sets the value of **handleInvalid**.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setInputCols(*value*) [source]
Sets the value of **inputCols**.

New in version 3.0.0.

setNumBuckets(*value*) [source]
Sets the value of **numBuckets**.

New in version 2.0.0.

setNumBucketsArray(*value*) [source]

Sets the value of **numBucketsArray**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setOutputCols(*value*) [source]

Sets the value of **outputCols**.

New in version 3.0.0.

setParams(*self*, *numBuckets=2*, *inputCol=None*, *outputCol=None*, *relativeError=0.001*, *handleInvalid="error"*, *numBucketsArray=None*, *inputCols=None*, *outputCols=None*) [source]

Set the params for the QuantileDiscretizer

New in version 2.0.0.

setRelativeError(*value*) [source]

Sets the value of **relativeError**.

New in version 2.0.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**RobustScaler**(*lower=0.25*, *upper=0.75*, *withCentering=False*, *withScaling=True*, *inputCol=None*, *outputCol=None*, *relativeError=0.001*) [source]

RobustScaler removes the median and scales the data according to the quantile range. The quantile range is by default IQR (Interquartile Range, quantile range between the 1st quartile = 25th quantile and the 3rd quartile = 75th quantile) but can be configured. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Median and quantile range are then stored to be used on later data using the transform method.

```
>>> from pyspark.ml.linalg import Vectors
>>> data = [(0, Vectors.dense([0.0, 0.0]),),
...         (1, Vectors.dense([1.0, -1.0]),),
...         (2, Vectors.dense([2.0, -2.0]),),
...         (3, Vectors.dense([3.0, -3.0]),),
...         (4, Vectors.dense([4.0, -4.0]),),]
>>> df = spark.createDataFrame(data, ["id", "features"])
>>> scaler = RobustScaler()
>>> scaler.setInputCol("features")
RobustScaler...
>>> scaler.setOutputCol("scaled")
RobustScaler...
>>> model = scaler.fit(df)
>>> model.setOutputCol("output")
RobustScalerModel...
>>> model.median
DenseVector([2.0, -2.0])
>>> model.range
DenseVector([2.0, 2.0])
>>> model.transform(df).collect()[1].output
DenseVector([0.5, -0.5])
>>> scalerPath = temp_path + "/robust-scaler"
>>> scaler.save(scalerPath)
>>> loadedScaler = RobustScaler.load(scalerPath)
>>> loadedScaler.getWithCentering() == scaler.getWithCentering()
True
>>> loadedScaler.getWithScaling() == scaler.getWithScaling()
True
>>> modelPath = temp_path + "/robust-scaler-model"
>>> model.save(modelPath)
>>> loadedModel = RobustScalerModel.load(modelPath)
>>> loadedModel.median == model.median
True
>>> loadedModel.range == model.range
True
```

New in version 3.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getLower()

Gets the value of lower or its default value.

New in version 3.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

getRelativeError()

Gets the value of relativeError or its default value.

getUpper()

Gets the value of upper or its default value.

New in version 3.0.0.

getWithCentering()

Gets the value of withCentering or its default value.

New in version 3.0.0.

getWithScaling()

Gets the value of withScaling or its default value.

New in version 3.0.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

lower = *Param(parent='undefined', name='lower', doc='Lower quantile to calculate quantile range')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

relativeError = *Param(parent='undefined', name='relativeError', doc='the relative target precision for the approximate quantile algorithm. Must be in the range [0, 1]')*

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setLower(*value*) [source]

Sets the value of **lower**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

setParams(*self*, *lower=0.25*, *upper=0.75*, *withCentering=False*, *withScaling=True*, *inputCol=None*, *outputCol=None*, *relativeError=0.001*) [source]

Sets params for this RobustScaler.

New in version 3.0.0.

setRelativeError(*value*) [source]

Sets the value of **relativeError**.

New in version 3.0.0.

setUpper(*value*) [source]

Sets the value of **upper**.

New in version 3.0.0.

setWithCentering(*value*) [source]

Sets the value of **withCentering**.

New in version 3.0.0.

setWithScaling(*value*) [source]

Sets the value of **withScaling**.

New in version 3.0.0.

upper = *Param(parent='undefined', name='upper', doc='Upper quantile to calculate quantile range')*

withCentering = *Param(parent='undefined', name='withCentering', doc='Whether to center data with median')*

withScaling = *Param(parent='undefined', name='withScaling', doc='Whether to scale the data to quantile range')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**RobustScalerModel**(*java_model=None*) [source]

Model fitted by **RobustScaler**.

New in version 3.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getLower()

Gets the value of lower or its default value.

New in version 3.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

getRelativeError()

Gets the value of relativeError or its default value.

getUpper()

Gets the value of upper or its default value.

New in version 3.0.0.

getWithCentering()

Gets the value of withCentering or its default value.

New in version 3.0.0.

getWithScaling()

Gets the value of withScaling or its default value.

New in version 3.0.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

lower = *Param(parent='undefined', name='lower', doc='Lower quantile to calculate quantile range')*

property **median**

Median of the RobustScalerModel.

New in version 3.0.0.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

property **range**

Quantile range of the RobustScalerModel.

New in version 3.0.0.

classmethod **read**()

Returns an MLReader instance for this class.

relativeError = *Param(parent='undefined', name='relativeError', doc='the relative target precision for the approximate quantile algorithm. Must be in the range [0, 1]')*

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

upper = *Param(parent='undefined', name='upper', doc='Upper quantile to calculate quantile range')*

withCentering = *Param(parent='undefined', name='withCentering', doc='Whether to center data with median')*

withScaling = *Param(parent='undefined', name='withScaling', doc='Whether to scale the data to quantile range')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**RegexTokenizer**(*minTokenLength=1*, *gaps=True*, *pattern='\s+'*, *inputCol=None*, *outputCol=None*, *toLowercase=True*) [source]

A regex based tokenizer that extracts tokens either by using the provided regex pattern (in Java dialect) to split the text (default) or repeatedly matching the regex (if gaps is false). Optional parameters also allow filtering tokens using a minimal length. It returns an array of strings that can be empty.

```
>>> df = spark.createDataFrame([("A B  c",)], ["text"])
>>> reTokenizer = RegexTokenizer()
>>> reTokenizer.setInputCol("text")
RegexTokenizer...
>>> reTokenizer.setOutputCol("words")
RegexTokenizer...
>>> reTokenizer.transform(df).head()
Row(text='A B  c', words=['a', 'b', 'c'])
>>> # Change a parameter.
>>> reTokenizer.setParams(outputCol="tokens").transform(df).head()
Row(text='A B  c', tokens=['a', 'b', 'c'])
>>> # Temporarily modify a parameter.
>>> reTokenizer.transform(df, {reTokenizer.outputCol: "words"}).head()
Row(text='A B  c', words=['a', 'b', 'c'])
>>> reTokenizer.transform(df).head()
Row(text='A B  c', tokens=['a', 'b', 'c'])
>>> # Must use keyword arguments to specify params.
>>> reTokenizer.setParams("text")
Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
>>> regexTokenizerPath = temp_path + "/regex-tokenizer"
>>> reTokenizer.save(regexTokenizerPath)
>>> loadedReTokenizer = RegexTokenizer.load(regexTokenizerPath)
>>> loadedReTokenizer.getMinTokenLength() == reTokenizer.getMinTokenLength()
True
>>> loadedReTokenizer.getGaps() == reTokenizer.getGaps()
True
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

gaps = *Param(parent='undefined', name='gaps', doc='whether regex splits on gaps (True) or matches tokens (False)')*

getGaps() [source]

Gets the value of gaps or its default value.

New in version 1.4.0.

getInputCol()

Gets the value of inputCol or its default value.

getMinTokenLength() [source]

Gets the value of minTokenLength or its default value.

New in version 1.4.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

getPattern() [source]

Gets the value of pattern or its default value.

New in version 1.4.0.

getToLowercase() [source]

Gets the value of toLowercase or its default value.

New in version 2.0.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

minTokenLength = *Param(parent='undefined', name='minTokenLength', doc='minimum token length (>= 0)')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

pattern = *Param(parent='undefined', name='pattern', doc='regex pattern (Java dialect) used for tokenizing')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setGaps(*value*) [source]

Sets the value of **gaps**.

New in version 1.4.0.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setMinTokenLength(*value*) [source]

Sets the value of **minTokenLength**.

New in version 1.4.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *minTokenLength=1*, *gaps=True*, *pattern="s+"*, *inputCol=None*, *outputCol=None*, *toLowercase=True*)

Sets params for this RegexTokenizer. [source]

New in version 1.4.0.

setPattern(*value*) [source]

Sets the value of **pattern**.

New in version 1.4.0.

setToLowercase(*value*) [source]

Sets the value of **toLowercase**.

New in version 2.0.0.

toLowercase = *Param(parent='undefined', name='toLowercase', doc='whether to convert all characters to lowercase before tokenizing')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**RFormula**(*formula=None*, *featuresCol='features'*, *labelCol='label'*, *forceIndexLabel=False*, *stringIndexerOrderType='frequencyDesc'*, *handleInvalid='error'*) [source]

Implements the transforms required for fitting a dataset against an R model formula. Currently we support a limited subset of the R operators, including '~', '.', ':', '+', '-', '*', and '^'. Also see the R formula docs.

```
>>> df = spark.createDataFrame([
...     (1.0, 1.0, "a"),
...     (0.0, 2.0, "b"),
...     (0.0, 0.0, "a")
... ], ["y", "x", "s"])
>>> rf = RFormula(formula="y ~ x + s")
>>> model = rf.fit(df)
>>> model.getLabelCol()
'label'
>>> model.transform(df).show()
+---+---+---+---------+-----+
|  y|  x|  s| features|label|
+---+---+---+---------+-----+
|1.0|1.0|  a|[1.0,1.0]|  1.0|
|0.0|2.0|  b|[2.0,0.0]|  0.0|
|0.0|0.0|  a|[0.0,1.0]|  0.0|
+---+---+---+---------+-----+
...
>>> rf.fit(df, {rf.formula: "y ~ . - s"}).transform(df).show()
+---+---+---+--------+-----+
|  y|  x|  s|features|label|
+---+---+---+--------+-----+
|1.0|1.0|  a|   [1.0]|  1.0|
|0.0|2.0|  b|   [2.0]|  0.0|
|0.0|0.0|  a|   [0.0]|  0.0|
+---+---+---+--------+-----+
...
>>> rFormulaPath = temp_path + "/rFormula"
>>> rf.save(rFormulaPath)
>>> loadedRF = RFormula.load(rFormulaPath)
>>> loadedRF.getFormula() == rf.getFormula()
True
>>> loadedRF.getFeaturesCol() == rf.getFeaturesCol()
True
>>> loadedRF.getLabelCol() == rf.getLabelCol()
True
>>> loadedRF.getHandleInvalid() == rf.getHandleInvalid()
True
>>> str(loadedRF)
'RFormula(y ~ x + s) (uid=...)'
>>> modelPath = temp_path + "/rFormulaModel"
>>> model.save(modelPath)
>>> loadedModel = RFormulaModel.load(modelPath)
>>> loadedModel.uid == model.uid
True
>>> loadedModel.transform(df).show()
+---+---+---+---------+-----+
|  y|  x|  s| features|label|
+---+---+---+---------+-----+
|1.0|1.0|  a|[1.0,1.0]|  1.0|
|0.0|2.0|  b|[2.0,0.0]|  0.0|
|0.0|0.0|  a|[0.0,1.0]|  0.0|
+---+---+---+---------+-----+
...
>>> str(loadedModel)
'RFormulaModel(ResolvedRFormula(label=y, terms=[x,s], hasIntercept=true)) (uid=...)'
```

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:

extra – Extra parameters to copy to the new instance

Returns:

Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

forceIndexLabel = *Param(parent='undefined', name='forceIndexLabel', doc='Force to index label whether it is numeric or string')*

formula = *Param(parent='undefined', name='formula', doc='R model formula')*

getFeaturesCol()

Gets the value of featuresCol or its default value.

getForceIndexLabel()

Gets the value of **forceIndexLabel**.

New in version 2.1.0.

getFormula()

Gets the value of **formula**.

New in version 1.5.0.

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getStringIndexerOrderType()

Gets the value of **stringIndexerOrderType** or its default value ‘frequencyDesc’.

New in version 2.3.0.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (put invalid data in a special additional bucket, at index numLabels).")*

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*) [source]

Sets the value of **featuresCol**.

setForceIndexLabel(*value*) [source]

Sets the value of **forceIndexLabel**.

New in version 2.1.0.

setFormula(*value*) [source]

Sets the value of **formula**.

New in version 1.5.0.

setHandleInvalid(*value*) [source]

Sets the value of **handleInvalid**.

setLabelCol(*value*) [source]

Sets the value of **labelCol**.

setParams(*self*, *formula=None*, *featuresCol="features"*, *labelCol="label"*, *forceIndexLabel=False*, *stringIndexerOrderType="frequencyDesc"*, *handleInvalid="error"*) [source]

Sets params for RFormula.

New in version 1.5.0.

setStringIndexerOrderType(*value*) [source]

Sets the value of **stringIndexerOrderType**.

New in version 2.3.0.

stringIndexerOrderType = *Param(parent='undefined', name='stringIndexerOrderType', doc='How to order categories of a string feature column used by StringIndexer. The last category after ordering is dropped when encoding strings. Supported options: frequencyDesc, frequencyAsc, alphabetDesc, alphabetAsc. The default value is frequencyDesc. When the ordering is set to alphabetDesc, RFormula drops the same category as R when encoding strings.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**RFormulaModel**(*java_model=None*) [source]

Model fitted by **RFormula**. Fitting is required to determine the factor levels of formula terms.

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

forceIndexLabel = *Param(parent='undefined', name='forceIndexLabel', doc='Force to index label whether it is numeric or string')*

formula = *Param(parent='undefined', name='formula', doc='R model formula')*

getFeaturesCol()

Gets the value of featuresCol or its default value.

getForceIndexLabel()

Gets the value of **forceIndexLabel**.

New in version 2.1.0.

getFormula()

Gets the value of **formula**.

New in version 1.5.0.

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getStringIndexerOrderType()

Gets the value of **stringIndexerOrderType** or its default value 'frequencyDesc'.

New in version 2.3.0.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="how to handle invalid entries. Options are 'skip' (filter out rows with invalid values), 'error' (throw an error), or 'keep' (put invalid data in a special additional bucket, at index numLabels).")*

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

stringIndexerOrderType = *Param(parent='undefined', name='stringIndexerOrderType', doc='How to order categories of a string feature column used by StringIndexer. The last category after ordering is dropped when encoding strings. Supported options: frequencyDesc, frequencyAsc, alphabetDesc, alphabetAsc. The default value is frequencyDesc. When the ordering is set to alphabetDesc, RFormula drops the same category as R when encoding strings.')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**SQLTransformer**(*statement=None*) [source]

Implements the transforms which are defined by SQL statement. Currently we only support SQL syntax like 'SELECT … FROM __THIS__' where '__THIS__' represents the underlying table of the input dataset.

```
>>> df = spark.createDataFrame([(0, 1.0, 3.0), (2, 2.0, 5.0)], ["id", "v1", "v2"])
>>> sqlTrans = SQLTransformer(
...     statement="SELECT *, (v1 + v2) AS v3, (v1 * v2) AS v4 FROM __THIS__")
>>> sqlTrans.transform(df).head()
Row(id=0, v1=1.0, v2=3.0, v3=4.0, v4=3.0)
>>> sqlTransformerPath = temp_path + "/sql-transformer"
>>> sqlTrans.save(sqlTransformerPath)
>>> loadedSqlTrans = SQLTransformer.load(sqlTransformerPath)
>>> loadedSqlTrans.getStatement() == sqlTrans.getStatement()
True
```

New in version 1.6.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:

extra – Extra parameters to copy to the new instance

Returns:

Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from

input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getStatement() [source]

Gets the value of statement or its default value.

New in version 1.6.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setParams(*self*, *statement=None*) [source]

Sets params for this SQLTransformer.

New in version 1.6.0.

setStatement(*value*) [source]

Sets the value of **statement**.

New in version 1.6.0.

statement = *Param(parent='undefined', name='statement', doc='SQL statement')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**StandardScaler**(*withMean=False*, *withStd=True*, *inputCol=None*, *outputCol=None*) [source]

Standardizes features by removing the mean and scaling to unit variance using column summary statistics on the samples in the training set.

The "unit std" is computed using the corrected sample standard deviation, which is computed as the square root of the unbiased sample variance.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(Vectors.dense([0.0]),), (Vectors.dense([2.0]),)], ["a"])
>>> standardScaler = StandardScaler()
>>> standardScaler.setInputCol("a")
StandardScaler...
>>> standardScaler.setOutputCol("scaled")
StandardScaler...
>>> model = standardScaler.fit(df)
>>> model.getInputCol()
'a'
>>> model.setOutputCol("output")
StandardScalerModel...
>>> model.mean
DenseVector([1.0])
>>> model.std
DenseVector([1.4142])
>>> model.transform(df).collect()[1].output
DenseVector([1.4142])
>>> standardScalerPath = temp_path + "/standard-scaler"
>>> standardScaler.save(standardScalerPath)
>>> loadedStandardScaler = StandardScaler.load(standardScalerPath)
>>> loadedStandardScaler.getWithMean() == standardScaler.getWithMean()
True
>>> loadedStandardScaler.getWithStd() == standardScaler.getWithStd()
True
>>> modelPath = temp_path + "/standard-scaler-model"
>>> model.save(modelPath)
>>> loadedModel = StandardScalerModel.load(modelPath)
>>> loadedModel.std == model.std
True
>>> loadedModel.mean == model.mean
True
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

getWithMean()

Gets the value of withMean or its default value.

New in version 1.4.0.

getWithStd()

Gets the value of withStd or its default value.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

setParams(*self*, *withMean=False*, *withStd=True*, *inputCol=None*, *outputCol=None*) [source]
Sets params for this StandardScaler.

New in version 1.4.0.

setWithMean(*value*) [source]
Sets the value of **withMean**.

New in version 1.4.0.

setWithStd(*value*) [source]
Sets the value of **withStd**.

New in version 1.4.0.

withMean = *Param(parent='undefined', name='withMean', doc='Center data with mean')*

withStd = *Param(parent='undefined', name='withStd', doc='Scale to unit standard deviation')*

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**StandardScalerModel**(*java_model=None*) [source]
Model fitted by **StandardScaler**.

New in version 1.4.0.

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()
Gets the value of inputCol or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

getWithMean()
Gets the value of withMean or its default value.

New in version 1.4.0.

getWithStd()
Gets the value of withStd or its default value.

New in version 1.4.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **mean**

Mean of the StandardScalerModel.

New in version 2.0.0.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

property **std**

Standard deviation of the StandardScalerModel.

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

withMean = *Param(parent='undefined', name='withMean', doc='Center data with mean')*

withStd = *Param(parent='undefined', name='withStd', doc='Scale to unit standard deviation')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**StopWordsRemover**(*inputCol=None*, *outputCol=None*, *stopWords=None*, *caseSensitive=False*, *locale=None*, *inputCols=None*, *outputCols=None*) [source]

A feature transformer that filters out stop words from input. Since 3.0.0, **StopWordsRemover** can filter out multiple columns at once by setting the **inputCols** parameter. Note that when both the **inputCol** and **inputCols** parameters are set, an Exception will be thrown.

Note: null values from input array are preserved unless adding null to stopWords explicitly.

```
>>> df = spark.createDataFrame([(["a", "b", "c"],)], ["text"])
>>> remover = StopWordsRemover(stopWords=["b"])
>>> remover.setInputCol("text")
StopWordsRemover...
>>> remover.setOutputCol("words")
StopWordsRemover...
>>> remover.transform(df).head().words == ['a', 'c']
True
>>> stopWordsRemoverPath = temp_path + "/stopwords-remover"
>>> remover.save(stopWordsRemoverPath)
>>> loadedRemover = StopWordsRemover.load(stopWordsRemoverPath)
>>> loadedRemover.getStopWords() == remover.getStopWords()
True
>>> loadedRemover.getCaseSensitive() == remover.getCaseSensitive()
True
>>> df2 = spark.createDataFrame([(["a", "b", "c"], ["a", "b"])], ["text1", "text2"])
>>> remover2 = StopWordsRemover(stopWords=["b"])
>>> remover2.setInputCols(["text1", "text2"]).setOutputCols(["words1", "words2"])
StopWordsRemover...
>>> remover2.transform(df2).show()
+---------+------+------+------+
|    text1| text2|words1|words2|
+---------+------+------+------+
|[a, b, c]|[a, b]|[a, c]|   [a]|
+---------+------+------+------+
...
```

New in version 1.6.0.

caseSensitive = *Param(parent='undefined', name='caseSensitive', doc='whether to do a case sensitive comparison over the stop words')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance

Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getCaseSensitive() [source]

Gets the value of **caseSensitive** or its default value.

New in version 1.6.0.

getInputCol()

Gets the value of inputCol or its default value.

getInputCols()

Gets the value of inputCols or its default value.

getLocale() [source]

Gets the value of **locale**.

New in version 2.4.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getOutputCols()

Gets the value of outputCols or its default value.

getParam(*paramName*)

Gets a param by its name.

getStopWords() [source]

Gets the value of **stopWords** or its default value.

New in version 1.6.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

static **loadDefaultStopWords**(*language*) [source]

Loads the default stop words for the given language. Supported languages: danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, portuguese, russian, spanish, swedish, turkish

New in version 2.0.0.

locale = *Param(parent='undefined', name='locale', doc='locale of the input. ignored when case sensitive is true')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setCaseSensitive(*value*) [source]

Sets the value of **caseSensitive**.

New in version 1.6.0.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setInputCols(*value*) [source]

Sets the value of **inputCols**.

New in version 3.0.0.

setLocale(*value*) [source]

Sets the value of **locale**.

New in version 2.4.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setOutputCols(*value*) [source]

Sets the value of **outputCols**.

New in version 3.0.0.

setParams(*self*, *inputCol=None*, *outputCol=None*, *stopWords=None*, *caseSensitive=false*, *locale=None*, *inputCols=None*, *outputCols=None*) [source]

Sets params for this StopWordRemover.

New in version 1.6.0.

setStopWords(*value*) [source]

Sets the value of **stopWords**.

New in version 1.6.0.

stopWords = *Param(parent='undefined', name='stopWords', doc='The words to be filtered out')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**StringIndexer**(*inputCol=None*, *outputCol=None*, *inputCols=None*, *outputCols=None*, *handleInvalid='error'*, *stringOrderType='frequencyDesc'*) [source]

A label indexer that maps a string column of labels to an ML column of label indices. If the input column is numeric, we cast it to string and index the string values. The indices are in [0, numLabels). By default, this is ordered by label frequencies so the most frequent label gets index 0. The ordering behavior is controlled by setting **stringOrderType**. Its default value is 'frequencyDesc'.

```
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed",
...     stringOrderType="frequencyDesc")
>>> stringIndexer.setHandleInvalid("error")
StringIndexer...
>>> model = stringIndexer.fit(stringIndDf)
>>> model.setHandleInvalid("error")
StringIndexerModel...
>>> td = model.transform(stringIndDf)
>>> sorted(set([(i[0], i[1]) for i in td.select(td.id, td.indexed).collect()]),
...     key=lambda x: x[0])
[(0, 0.0), (1, 2.0), (2, 1.0), (3, 0.0), (4, 0.0), (5, 1.0)]
>>> inverter = IndexToString(inputCol="indexed", outputCol="label2", labels=model.labels)
>>> itd = inverter.transform(td)
>>> sorted(set([(i[0], str(i[1])) for i in itd.select(itd.id, itd.label2).collect()]),
...     key=lambda x: x[0])
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'a'), (4, 'a'), (5, 'c')]
>>> stringIndexerPath = temp_path + "/string-indexer"
>>> stringIndexer.save(stringIndexerPath)
>>> loadedIndexer = StringIndexer.load(stringIndexerPath)
>>> loadedIndexer.getHandleInvalid() == stringIndexer.getHandleInvalid()
True
>>> modelPath = temp_path + "/string-indexer-model"
>>> model.save(modelPath)
>>> loadedModel = StringIndexerModel.load(modelPath)
>>> loadedModel.labels == model.labels
True
>>> indexToStringPath = temp_path + "/index-to-string"
>>> inverter.save(indexToStringPath)
>>> loadedInverter = IndexToString.load(indexToStringPath)
>>> loadedInverter.getLabels() == inverter.getLabels()
True
>>> stringIndexer.getStringOrderType()
'frequencyDesc'
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed", handleInvalid="error",
...     stringOrderType="alphabetDesc")
>>> model = stringIndexer.fit(stringIndDf)
>>> td = model.transform(stringIndDf)
>>> sorted(set([(i[0], i[1]) for i in td.select(td.id, td.indexed).collect()]),
...     key=lambda x: x[0])
[(0, 2.0), (1, 1.0), (2, 0.0), (3, 2.0), (4, 2.0), (5, 0.0)]
>>> fromlabelsModel = StringIndexerModel.from_labels(["a", "b", "c"],
...     inputCol="label", outputCol="indexed", handleInvalid="error")
>>> result = fromlabelsModel.transform(stringIndDf)
>>> sorted(set([(i[0], i[1]) for i in result.select(result.id, result.indexed).collect()]),
...     key=lambda x: x[0])
[(0, 0.0), (1, 1.0), (2, 2.0), (3, 0.0), (4, 0.0), (5, 2.0)]
>>> testData = sc.parallelize([Row(id=0, label1="a", label2="e"),
...                            Row(id=1, label1="b", label2="f"),
...                            Row(id=2, label1="c", label2="e"),
...                            Row(id=3, label1="a", label2="f"),
...                            Row(id=4, label1="a", label2="f"),
...                            Row(id=5, label1="c", label2="f")], 3)
>>> multiRowDf = spark.createDataFrame(testData)
>>> inputs = ["label1", "label2"]
>>> outputs = ["index1", "index2"]
>>> stringIndexer = StringIndexer(inputCols=inputs, outputCols=outputs)
>>> model = stringIndexer.fit(multiRowDf)
>>> result = model.transform(multiRowDf)
>>> sorted(set([(i[0], i[1], i[2]) for i in result.select(result.id, result.index1,
...     result.index2).collect()]), key=lambda x: x[0])
[(0, 0.0, 1.0), (1, 2.0, 0.0), (2, 1.0, 1.0), (3, 0.0, 0.0), (4, 0.0, 0.0), (5, 1.0, 0.0)]
>>> fromlabelsModel = StringIndexerModel.from_arrays_of_labels([["a", "b", "c"], ["e", "f"]],
...     inputCols=inputs, outputCols=outputs)
>>> result = fromlabelsModel.transform(multiRowDf)
>>> sorted(set([(i[0], i[1], i[2]) for i in result.select(result.id, result.index1,
...     result.index2).collect()]), key=lambda x: x[0])
[(0, 0.0, 0.0), (1, 1.0, 1.0), (2, 2.0, 0.0), (3, 0.0, 1.0), (4, 0.0, 1.0), (5, 2.0, 1.0)]
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)
Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)
Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getHandleInvalid()
Gets the value of handleInvalid or its default value.

getInputCol()
Gets the value of inputCol or its default value.

getInputCols()
Gets the value of inputCols or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getOutputCols()
Gets the value of outputCols or its default value.

getParam(*paramName*)
Gets a param by its name.

getStringOrderType()
Gets the value of **stringOrderType** or its default value ‘frequencyDesc’.

New in version 2.3.0.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="how to handle invalid data (unseen or NULL values) in features and label column of string type. Options are 'skip' (filter out rows with invalid data), error (throw an error), or 'keep' (put invalid data in a special additional bucket, at index numLabels).")*

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setHandleInvalid(*value*) [source]

Sets the value of **handleInvalid**.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setInputCols(*value*) [source]

Sets the value of **inputCols**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setOutputCols(*value*) [source]

Sets the value of **outputCols**.

New in version 3.0.0.

setParams(*self*, *inputCol=None*, *outputCol=None*, *inputCols=None*, *outputCols=None*, *handleInvalid="error"*, *stringOrderType="frequencyDesc"*) [source]

Sets params for this StringIndexer.

New in version 1.4.0.

setStringOrderType(*value*) [source]

Sets the value of **stringOrderType**.

New in version 2.3.0.

stringOrderType = *Param(parent='undefined', name='stringOrderType', doc='How to order labels of string column. The first label after ordering is assigned an index of 0. Supported options: frequencyDesc, frequencyAsc, alphabetDesc, alphabetAsc. Default is frequencyDesc. In case of equal frequency when under frequencyDesc/Asc, the strings are further sorted alphabetically')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**StringIndexerModel**(*java_model=None*) [source]

Model fitted by **StringIndexer**.

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

classmethod **from_arrays_of_labels**(*arrayOfLabels*, *inputCols*, *outputCols=None*, *handleInvalid=None*) [source]

Construct the model directly from an array of array of label strings, requires an active SparkContext.

New in version 3.0.0.

classmethod **from_labels**(*labels*, *inputCol*, *outputCol=None*, *handleInvalid=None*) [source]

Construct the model directly from an array of label strings, requires an active SparkContext.

New in version 2.4.0.

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getInputCol()

Gets the value of inputCol or its default value.

getInputCols()

Gets the value of inputCols or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getOutputCols()
Gets the value of outputCols or its default value.

getParam(*paramName*)
Gets a param by its name.

getStringOrderType()
Gets the value of **stringOrderType** or its default value 'frequencyDesc'.

New in version 2.3.0.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="how to handle invalid data (unseen or NULL values) in features and label column of string type. Options are 'skip' (filter out rows with invalid data), error (throw an error), or 'keep' (put invalid data in a special additional bucket, at index numLabels).")*

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

property **labels**
Ordered list of labels, corresponding to indices to be assigned.

New in version 1.5.0.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

outputCols = *Param(parent='undefined', name='outputCols', doc='output column names.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setHandleInvalid(*value*) [source]
Sets the value of **handleInvalid**.

New in version 2.4.0.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setInputCols(*value*) [source]
Sets the value of **inputCols**.

New in version 3.0.0.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

setOutputCols(*value*) [source]
Sets the value of **outputCols**.

New in version 3.0.0.

stringOrderType = *Param(parent='undefined', name='stringOrderType', doc='How to order labels of string column. The first label after ordering is assigned an index of 0. Supported options: frequencyDesc, frequencyAsc, alphabetDesc, alphabetAsc. Default is frequencyDesc. In case of equal frequency when under frequencyDesc/Asc, the strings are further sorted alphabetically')*

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**Tokenizer**(*inputCol=None*, *outputCol=None*) [source]
A tokenizer that converts the input string to lowercase and then splits it by white spaces.

```
>>> df = spark.createDataFrame([("a b c",)], ["text"])
>>> tokenizer = Tokenizer(outputCol="words")
>>> tokenizer.setInputCol("text")
Tokenizer...
>>> tokenizer.transform(df).head()
Row(text='a b c', words=['a', 'b', 'c'])
>>> # Change a parameter.
>>> tokenizer.setParams(outputCol="tokens").transform(df).head()
Row(text='a b c', tokens=['a', 'b', 'c'])
>>> # Temporarily modify a parameter.
>>> tokenizer.transform(df, {tokenizer.outputCol: "words"}).head()
Row(text='a b c', words=['a', 'b', 'c'])
>>> tokenizer.transform(df).head()
Row(text='a b c', tokens=['a', 'b', 'c'])
>>> # Must use keyword arguments to specify params.
>>> tokenizer.setParams("text")
Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
>>> tokenizerPath = temp_path + "/tokenizer"
>>> tokenizer.save(tokenizerPath)
>>> loadedTokenizer = Tokenizer.load(tokenizerPath)
>>> loadedTokenizer.transform(df).head().tokens == tokenizer.transform(df).head().tokens
True
```

New in version 1.3.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getInputCol()

Gets the value of inputCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this Tokenizer.

New in version 1.3.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**VectorAssembler**(*inputCols=None*, *outputCol=None*, *handleInvalid='error'*) [source]

A feature transformer that merges multiple columns into a vector column.

```
>>> df = spark.createDataFrame([(1, 0, 3)], ["a", "b", "c"])
>>> vecAssembler = VectorAssembler(outputCol="features")
>>> vecAssembler.setInputCols(["a", "b", "c"])
VectorAssembler...
>>> vecAssembler.transform(df).head().features
DenseVector([1.0, 0.0, 3.0])
>>> vecAssembler.setParams(outputCol="freqs").transform(df).head().freqs
DenseVector([1.0, 0.0, 3.0])
>>> params = {vecAssembler.inputCols: ["b", "a"], vecAssembler.outputCol: "vector"}
>>> vecAssembler.transform(df, params).head().vector
DenseVector([0.0, 1.0])
>>> vectorAssemblerPath = temp_path + "/vector-assembler"
>>> vecAssembler.save(vectorAssemblerPath)
>>> loadedAssembler = VectorAssembler.load(vectorAssemblerPath)
>>> loadedAssembler.transform(df).head().freqs == vecAssembler.transform(df).head().freqs
True
>>> dfWithNullsAndNaNs = spark.createDataFrame(
...     [(1.0, 2.0, None), (3.0, float("nan"), 4.0), (5.0, 6.0, 7.0)], ["a", "b", "c"])
>>> vecAssembler2 = VectorAssembler(inputCols=["a", "b", "c"], outputCol="features",
...     handleInvalid="keep")
>>> vecAssembler2.transform(dfWithNullsAndNaNs).show()
+---+---+----+-------------+
|  a|  b|   c|     features|
+---+---+----+-------------+
|1.0|2.0|null|[1.0,2.0,NaN]|
|3.0|NaN| 4.0|[3.0,NaN,4.0]|
|5.0|6.0| 7.0|[5.0,6.0,7.0]|
+---+---+----+-------------+
...
>>> vecAssembler2.setParams(handleInvalid="skip").transform(dfWithNullsAndNaNs).show()
+---+---+---+-------------+
|  a|  b|  c|     features|
+---+---+---+-------------+
|5.0|6.0|7.0|[5.0,6.0,7.0]|
+---+---+---+-------------+
...
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getInputCols()

Gets the value of inputCols or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="How to handle invalid data (NULL and NaN values). Options are 'skip' (filter out rows with invalid data), 'error' (throw an error), or 'keep' (return relevant number of NaN in the output). Column lengths are taken from the size of ML Attribute Group, which can be set using `VectorSizeHint` in a pipeline before `VectorAssembler`. Column lengths can also be inferred from first rows of the data since it is safe to do so but only in case of 'error' or 'skip').")*

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCols = *Param(parent='undefined', name='inputCols', doc='input column names.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setHandleInvalid(*value*) [source]

Sets the value of **handleInvalid**.

setInputCols(*value*) [source]

Sets the value of **inputCols**.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *inputCols=None*, *outputCol=None*, *handleInvalid="error"*) [source]

Sets params for this VectorAssembler.

New in version 1.4.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**VectorIndexer**(*maxCategories=20*, *inputCol=None*, *outputCol=None*, *handleInvalid='error'*) [source]

Class for indexing categorical feature columns in a dataset of *Vector*.

This has 2 usage modes:

- Automatically identify categorical features (default behavior)
  - This helps process a dataset of unknown vectors into a dataset with some continuous features and some categorical features. The choice between continuous and categorical is based upon a maxCategories parameter.
  - Set maxCategories to the maximum number of categorical any categorical feature should have.
  - E.g.: Feature 0 has unique values {-1.0, 0.0}, and feature 1 values {1.0, 3.0, 5.0}. If maxCategories = 2, then feature 0 will be declared categorical and use indices {0, 1}, and feature 1 will be declared continuous.
- Index all features, if all features are categorical
  - If maxCategories is set to be very large, then this will build an index of unique values for all features.
  - Warning: This can cause problems if features are continuous since this will collect ALL unique values to the driver.
  - E.g.: Feature 0 has unique values {-1.0, 0.0}, and feature 1 values {1.0, 3.0, 5.0}. If maxCategories >= 3, then both features will be declared categorical.

This returns a model which can transform categorical features to use 0-based indices.

Index stability:

- This is not guaranteed to choose the same category index across multiple runs.
- If a categorical feature includes value 0, then this is guaranteed to map value 0 to index 0. This maintains vector sparsity.
- More stability may be added in the future.

TODO: Future extensions: The following functionality is planned for the future:

- Preserve metadata in transform; if a feature's metadata is already present, do not recompute.
- Specify certain features to not index, either via a parameter or via existing metadata.
- Add warning if a categorical feature has only 1 category.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([(Vectors.dense([-1.0, 0.0]),),
...     (Vectors.dense([0.0, 1.0]),), (Vectors.dense([0.0, 2.0]),)], ["a"])
>>> indexer = VectorIndexer(maxCategories=2, inputCol="a")
>>> indexer.setOutputCol("indexed")
VectorIndexer...
>>> model = indexer.fit(df)
>>> indexer.getHandleInvalid()
'error'
>>> model.setOutputCol("output")
VectorIndexerModel...
>>> model.transform(df).head().output
DenseVector([1.0, 0.0])
>>> model.numFeatures
2
>>> model.categoryMaps
{0: {0.0: 0, -1.0: 1}}
>>> indexer.setParams(outputCol="test").fit(df).transform(df).collect()[1].test
DenseVector([0.0, 1.0])
>>> params = {indexer.maxCategories: 3, indexer.outputCol: "vector"}
>>> model2 = indexer.fit(df, params)
>>> model2.transform(df).head().vector
DenseVector([1.0, 0.0])
>>> vectorIndexerPath = temp_path + "/vector-indexer"
>>> indexer.save(vectorIndexerPath)
>>> loadedIndexer = VectorIndexer.load(vectorIndexerPath)
>>> loadedIndexer.getMaxCategories() == indexer.getMaxCategories()
True
>>> modelPath = temp_path + "/vector-indexer-model"
>>> model.save(modelPath)
>>> loadedModel = VectorIndexerModel.load(modelPath)
>>> loadedModel.numFeatures == model.numFeatures
True
>>> loadedModel.categoryMaps == model.categoryMaps
True
>>> dfWithInvalid = spark.createDataFrame([(Vectors.dense([3.0, 1.0]),)], ["a"])
>>> indexer.getHandleInvalid()
'error'
>>> model3 = indexer.setHandleInvalid("skip").fit(df)
>>> model3.transform(dfWithInvalid).count()
0
>>> model4 = indexer.setParams(handleInvalid="keep", outputCol="indexed").fit(df)
>>> model4.transform(dfWithInvalid).head().indexed
DenseVector([2.0, 1.0])
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getInputCol()

Gets the value of inputCol or its default value.

getMaxCategories()

Gets the value of maxCategories or its default value.

New in version 1.4.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="How to handle invalid data (unseen labels or NULL values). Options are 'skip' (filter out rows with invalid data), 'error' (throw an error), or 'keep' (put invalid data in a special additional bucket, at index of the number of categories of the feature).")*

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxCategories = *Param(parent='undefined', name='maxCategories', doc='Threshold for the number of values a categorical feature can take (>= 2). If a feature is found to have > maxCategories values, then it is declared continuous.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setHandleInvalid(*value*) [source]
Sets the value of **handleInvalid**.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setMaxCategories(*value*) [source]
Sets the value of **maxCategories**.

New in version 1.4.0.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

setParams(*self*, *maxCategories=20*, *inputCol=None*, *outputCol=None*, *handleInvalid="error"*) [source]
Sets params for this VectorIndexer.

New in version 1.4.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**VectorIndexerModel**(*java_model=None*) [source]
Model fitted by **VectorIndexer**.

Transform categorical features to use 0-based indices instead of their original values.

- Categorical features are mapped to indices.
- Continuous features (columns) are left unchanged.

This also appends metadata to the output column, marking features as Numeric (continuous), Nominal (categorical), or Binary (either continuous or categorical). Non-ML metadata is not carried over from the input to the output column.

This maintains vector sparsity.

New in version 1.4.0.

property **categoryMaps**
Feature value index. Keys are categorical feature indices (column indices). Values are maps from original features values to 0-based category indices. If a feature is not in this map, it is treated as continuous.

New in version 1.4.0.

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getInputCol()

Gets the value of inputCol or its default value.

getMaxCategories()

Gets the value of maxCategories or its default value.

New in version 1.4.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc="How to handle invalid data (unseen labels or NULL values). Options are 'skip' (filter out rows with invalid data), 'error' (throw an error), or 'keep' (put invalid data in a special additional bucket, at index of the number of categories of the feature).")*

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxCategories = *Param(parent='undefined', name='maxCategories', doc='Threshold for the number of values a categorical feature can take (>= 2). If a feature is found to have > maxCategories values, then it is declared continuous.')*

property **numFeatures**

Number of features, i.e., length of Vectors which this transforms.

New in version 1.4.0.

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

New in version 3.0.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**VectorSizeHint**(*inputCol=None*, *size=None*, *handleInvalid='error'*) [source]

A feature transformer that adds size information to the metadata of a vector column. VectorAssembler needs size

information for its input columns and cannot be used on streaming dataframes without this metadata.

Note: VectorSizeHint modifies *inputCol* to include size metadata and does not have an outputCol.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml import Pipeline, PipelineModel
>>> data = [(Vectors.dense([1., 2., 3.]), 4.)]
>>> df = spark.createDataFrame(data, ["vector", "float"])
>>>
>>> sizeHint = VectorSizeHint(inputCol="vector", size=3, handleInvalid="skip")
>>> vecAssembler = VectorAssembler(inputCols=["vector", "float"], outputCol="assembled")
>>> pipeline = Pipeline(stages=[sizeHint, vecAssembler])
>>>
>>> pipelineModel = pipeline.fit(df)
>>> pipelineModel.transform(df).head().assembled
DenseVector([1.0, 2.0, 3.0, 4.0])
>>> vectorSizeHintPath = temp_path + "/vector-size-hint-pipeline"
>>> pipelineModel.save(vectorSizeHintPath)
>>> loadedPipeline = PipelineModel.load(vectorSizeHintPath)
>>> loaded = loadedPipeline.transform(df).head().assembled
>>> expected = pipelineModel.transform(df).head().assembled
>>> loaded == expected
True
```

New in version 2.3.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getHandleInvalid()

Gets the value of handleInvalid or its default value.

getInputCol()

Gets the value of inputCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getSize() [source]

Gets size param, the size of vectors in *inputCol*.

New in version 2.3.0.

handleInvalid = *Param(parent='undefined', name='handleInvalid', doc='How to handle invalid vectors in inputCol. Invalid vectors include nulls and vectors with the wrong size. The options are `skip` (filter out rows with invalid vectors), `error` (throw an error) and `optimistic` (do not check the vector size, and keep all rows). `error` by default.')*

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setHandleInvalid(*value*) [source]

Sets the value of **handleInvalid**.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setParams(*self*, *inputCol=None*, *size=None*, *handleInvalid="error"*) [source]

Sets params for this VectorSizeHint.

New in version 2.3.0.

setSize(*value*) [source]

Sets size param, the size of vectors in *inputCol*.

New in version 2.3.0.

size = *Param(parent='undefined', name='size', doc='Size of vectors in column.')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**VectorSlicer**(*inputCol=None*, *outputCol=None*, *indices=None*, *names=None*) [source]

This class takes a feature vector and outputs a new feature vector with a subarray of the original features.

The subset of features can be specified with either indices (*setIndices()*) or names (*setNames()*). At least one feature must be selected. Duplicate features are not allowed, so there can be no overlap between selected indices and names.

The output vector will order features with the selected indices first (in the order given), followed by the selected names (in the order given).

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (Vectors.dense([-2.0, 2.3, 0.0, 0.0, 1.0]),),
...     (Vectors.dense([0.0, 0.0, 0.0, 0.0, 0.0]),),
...     (Vectors.dense([0.6, -1.1, -3.0, 4.5, 3.3]),)], ["features"])
>>> vs = VectorSlicer(outputCol="sliced", indices=[1, 4])
>>> vs.setInputCol("features")
VectorSlicer...
>>> vs.transform(df).head().sliced
DenseVector([2.3, 1.0])
>>> vectorSlicerPath = temp_path + "/vector-slicer"
>>> vs.save(vectorSlicerPath)
>>> loadedVs = VectorSlicer.load(vectorSlicerPath)
>>> loadedVs.getIndices() == vs.getIndices()
True
>>> loadedVs.getNames() == vs.getNames()
True
```

New in version 1.6.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getIndices() [source]

Gets the value of indices or its default value.

New in version 1.6.0.

getInputCol()

Gets the value of inputCol or its default value.

getNames() [source]

Gets the value of names or its default value.

New in version 1.6.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

indices = *Param(parent='undefined', name='indices', doc='An array of indices to select features from a vector column. There can be no overlap with names.')*

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

names = *Param(parent='undefined', name='names', doc='An array of feature names to select features from a vector column. These names must be specified by ML org.apache.spark.ml.attribute.Attribute. There can be no overlap with indices.')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setIndices(*value*) [source]

Sets the value of **indices**.

New in version 1.6.0.

setInputCol(*value*) [source]

Sets the value of **inputCol**.

setNames(*value*) [source]

Sets the value of **names**.

New in version 1.6.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*inputCol=None*, *outputCol=None*, *indices=None*, *names=None*) [source]

setParams(self, inputCol=None, outputCol=None, indices=None, names=None): Sets params for this VectorSlicer.

New in version 1.6.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**Word2Vec**(*vectorSize=100*, *minCount=5*, *numPartitions=1*, *stepSize=0.025*, *maxIter=1*, *seed=None*, *inputCol=None*, *outputCol=None*, *windowSize=5*, *maxSentenceLength=1000*) [source]

Word2Vec trains a model of *Map(String, Vector)*, i.e. transforms a word into a code for further natural language processing or machine learning process.

```
>>> sent = ("a b " * 100 + "a c " * 10).split(" ")
>>> doc = spark.createDataFrame([(sent,), (sent,)], ["sentence"])
>>> word2Vec = Word2Vec(vectorSize=5, seed=42, inputCol="sentence", outputCol="model")
>>> word2Vec.setMaxIter(10)
Word2Vec...
>>> word2Vec.getMaxIter()
10
>>> word2Vec.clear(word2Vec.maxIter)
>>> model = word2Vec.fit(doc)
>>> model.getMinCount()
5
>>> model.setInputCol("sentence")
Word2VecModel...
>>> model.getVectors().show()
+----+--------------------+
|word|              vector|
+----+--------------------+
|   a|[0.09511678665876...|
|   b|[-1.2028766870498...|
|   c|[0.30153277516365...|
+----+--------------------+
...
>>> model.findSynonymsArray("a", 2)
[('b', 0.015859870240092278), ('c', -0.5680795907974243)]
>>> from pyspark.sql.functions import format_number as fmt
>>> model.findSynonyms("a", 2).select("word", fmt("similarity", 5).alias("similarity")).show()
+----+----------+
|word|similarity|
+----+----------+
|   b|   0.01586|
|   c|  -0.56808|
+----+----------+
...
>>> model.transform(doc).head().model
DenseVector([-0.4833, 0.1855, -0.273, -0.0509, -0.4769])
>>> word2vecPath = temp_path + "/word2vec"
>>> word2Vec.save(word2vecPath)
>>> loadedWord2Vec = Word2Vec.load(word2vecPath)
>>> loadedWord2Vec.getVectorSize() == word2Vec.getVectorSize()
True
>>> loadedWord2Vec.getNumPartitions() == word2Vec.getNumPartitions()
True
>>> loadedWord2Vec.getMinCount() == word2Vec.getMinCount()
True
>>> modelPath = temp_path + "/word2vec-model"
>>> model.save(modelPath)
>>> loadedModel = Word2VecModel.load(modelPath)
>>> loadedModel.getVectors().first().word == model.getVectors().first().word
True
>>> loadedModel.getVectors().first().vector == model.getVectors().first().vector
True
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getMaxIter()

Gets the value of maxIter or its default value.

getMaxSentenceLength()
Gets the value of maxSentenceLength or its default value.

New in version 2.0.0.

getMinCount()
Gets the value of minCount or its default value.

New in version 1.4.0.

getNumPartitions()
Gets the value of numPartitions or its default value.

New in version 1.4.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

getSeed()
Gets the value of seed or its default value.

getStepSize()
Gets the value of stepSize or its default value.

getVectorSize()
Gets the value of vectorSize or its default value.

New in version 1.4.0.

getWindowSize()
Gets the value of windowSize or its default value.

New in version 2.0.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

maxSentenceLength = *Param(parent='undefined', name='maxSentenceLength', doc='Maximum length (in words) of each sentence in the input data. Any sentence longer than this threshold will be divided into chunks up to the size.')*

minCount = *Param(parent='undefined', name='minCount', doc="the minimum number of times a token must appear to be included in the word2vec model's vocabulary")*

numPartitions = *Param(parent='undefined', name='numPartitions', doc='number of partitions for sentences of words')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setMaxIter(*value*) [source]
Sets the value of **maxIter**.

setMaxSentenceLength(*value*) [source]
Sets the value of **maxSentenceLength**.

New in version 2.0.0.

setMinCount(*value*) [source]
Sets the value of **minCount**.

New in version 1.4.0.

setNumPartitions(*value*) [source]
Sets the value of **numPartitions**.

New in version 1.4.0.

setOutputCol(*value*) [source]

Sets the value of **outputCol**.

setParams(*self*, *minCount=5*, *numPartitions=1*, *stepSize=0.025*, *maxIter=1*, *seed=None*, *inputCol=None*, *outputCol=None*, *windowSize=5*, *maxSentenceLength=1000*) [source]

Sets params for this Word2Vec.

New in version 1.4.0.

setSeed(*value*) [source]

Sets the value of **seed**.

setStepSize(*value*) [source]

Sets the value of **stepSize**.

New in version 1.4.0.

setVectorSize(*value*) [source]

Sets the value of **vectorSize**.

New in version 1.4.0.

setWindowSize(*value*) [source]

Sets the value of **windowSize**.

New in version 2.0.0.

stepSize = *Param(parent='undefined', name='stepSize', doc='Step size to be used for each iteration of optimization (>= 0).')*

vectorSize = *Param(parent='undefined', name='vectorSize', doc='the dimension of codes after transforming from words')*

windowSize = *Param(parent='undefined', name='windowSize', doc='the window size (context words from [-window, window]). Default value is 5')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.feature.**Word2VecModel**(*java_model=None*) [source]

Model fitted by **Word2Vec**.

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

findSynonyms(*word*, *num*) [source]

Find “num” number of words closest in similarity to “word”. word can be a string or vector representation. Returns a dataframe with two fields word and similarity (which gives the cosine similarity).

New in version 1.5.0.

findSynonymsArray(*word*, *num*) [source]

Find “num” number of words closest in similarity to “word”. word can be a string or vector representation. Returns an array with two fields word and similarity (which gives the cosine similarity).

New in version 2.3.0.

getInputCol()

Gets the value of inputCol or its default value.

getMaxIter()

Gets the value of maxIter or its default value.

getMaxSentenceLength()

Gets the value of maxSentenceLength or its default value.

New in version 2.0.0.

getMinCount()

Gets the value of minCount or its default value.

New in version 1.4.0.

getNumPartitions()

Gets the value of numPartitions or its default value.

New in version 1.4.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getOutputCol()
Gets the value of outputCol or its default value.

getParam(*paramName*)
Gets a param by its name.

getSeed()
Gets the value of seed or its default value.

getStepSize()
Gets the value of stepSize or its default value.

getVectorSize()
Gets the value of vectorSize or its default value.

New in version 1.4.0.

getVectors() [source]
Returns the vector representation of the words as a dataframe with two fields, word and vector.

New in version 1.5.0.

getWindowSize()
Gets the value of windowSize or its default value.

New in version 2.0.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

inputCol = *Param(parent='undefined', name='inputCol', doc='input column name.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

maxSentenceLength = *Param(parent='undefined', name='maxSentenceLength', doc='Maximum length (in words) of each sentence in the input data. Any sentence longer than this threshold will be divided into chunks up to the size.')*

minCount = *Param(parent='undefined', name='minCount', doc="the minimum number of times a token must appear to be included in the word2vec model's vocabulary")*

numPartitions = *Param(parent='undefined', name='numPartitions', doc='number of partitions for sentences of words')*

outputCol = *Param(parent='undefined', name='outputCol', doc='output column name.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setInputCol(*value*) [source]
Sets the value of **inputCol**.

setOutputCol(*value*) [source]
Sets the value of **outputCol**.

stepSize = *Param(parent='undefined', name='stepSize', doc='Step size to be used for each iteration of optimization (>= 0).')*

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

vectorSize = *Param(parent='undefined', name='vectorSize', doc='the dimension of codes after transforming from words')*

windowSize = *Param(parent='undefined', name='windowSize', doc='the window size (context words from [-window, window]). Default value is 5')*

write()
Returns an MLWriter instance for this ML instance.

pyspark.ml.classification module

class `pyspark.ml.classification.`**`LinearSVC`**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction', fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2*) [source]

Linear SVM Classifier

This binary classifier optimizes the Hinge Loss using the OWLQN optimizer. Only supports L2 regularization currently.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = sc.parallelize([
...     Row(label=1.0, features=Vectors.dense(1.0, 1.0, 1.0)),
...     Row(label=0.0, features=Vectors.dense(1.0, 2.0, 3.0))]).toDF()
>>> svm = LinearSVC()
>>> svm.getMaxIter()
100
>>> svm.setMaxIter(5)
LinearSVC...
>>> svm.getMaxIter()
5
>>> svm.getRegParam()
0.0
>>> svm.setRegParam(0.01)
LinearSVC...
>>> svm.getRegParam()
0.01
>>> model = svm.fit(df)
>>> model.setPredictionCol("newPrediction")
LinearSVCModel...
>>> model.getPredictionCol()
'newPrediction'
>>> model.setThreshold(0.5)
LinearSVCModel...
>>> model.getThreshold()
0.5
>>> model.coefficients
DenseVector([0.0, -0.2792, -0.1833])
>>> model.intercept
1.0206118982229047
>>> model.numClasses
2
>>> model.numFeatures
3
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, -1.0, -1.0))]).toDF()
>>> model.predict(test0.head().features)
1.0
>>> result = model.transform(test0).head()
>>> result.newPrediction
1.0
>>> result.rawPrediction
DenseVector([-1.4831, 1.4831])
>>> svm_path = temp_path + "/svm"
>>> svm.save(svm_path)
>>> svm2 = LinearSVC.load(svm_path)
>>> svm2.getMaxIter()
5
>>> model_path = temp_path + "/svm_model"
>>> model.save(model_path)
>>> model2 = LinearSVCModel.load(model_path)
>>> model.coefficients[0] == model2.coefficients[0]
True
>>> model.intercept == model2.intercept
True
```

New in version 2.2.0.

`aggregationDepth` = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

`clear`(*param*)

Clears a param from the param map if it has been explicitly set.

`copy`(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

`explainParam`(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

`explainParams`()

Returns the documentation of all params with their optionally default values and user-supplied values.

`extractParamMap`(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

`featuresCol` = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

`fit`(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **`pyspark.sql.DataFrame`**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

`fitIntercept` = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

`fitMultiple`(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:

- **dataset** – input dataset, which is an instance of `pyspark.sql.DataFrame`.
- **paramMaps** – A Sequence of param maps.

Returns:

A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getAggregationDepth()

Gets the value of aggregationDepth or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getFitIntercept()

Gets the value of fitIntercept or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getRawPredictionCol()

Gets the value of rawPredictionCol or its default value.

getRegParam()

Gets the value of regParam or its default value.

getStandardization()

Gets the value of standardization or its default value.

getThreshold()

Gets the value of threshold or its default value.

getTol()

Gets the value of tol or its default value.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**

Returns all params ordered by name. The default implementation uses `dir()` to get all attributes of type `Param`.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setAggregationDepth(*value*) [source]

Sets the value of `aggregationDepth`.

New in version 2.2.0.

setFeaturesCol(*value*)

Sets the value of `featuresCol`.

New in version 3.0.0.

setFitIntercept(*value*) [source]

Sets the value of **fitIntercept**.

New in version 2.2.0.

setLabelCol(*value*)

Sets the value of **labelCol**.

New in version 3.0.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

New in version 2.2.0.

setParams(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction', fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2*) [source]

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, regParam=0.0, tol=1e-6, rawPredictionCol="rawPrediction", fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2): Sets params for Linear SVM Classifier.

New in version 2.2.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setRegParam(*value*) [source]

Sets the value of **regParam**.

New in version 2.2.0.

setStandardization(*value*) [source]

Sets the value of **standardization**.

New in version 2.2.0.

setThreshold(*value*) [source]

Sets the value of **threshold**.

New in version 2.2.0.

setTol(*value*) [source]

Sets the value of **tol**.

New in version 2.2.0.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

New in version 2.2.0.

standardization = *Param(parent='undefined', name='standardization', doc='whether to standardize the training features before fitting the model.')*

threshold = *Param(parent='undefined', name='threshold', doc='The threshold in binary classification applied to the linear model prediction. This threshold can be any real number, where Inf will make all predictions 0.0 and -Inf will make all predictions 1.0.')*

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**LinearSVCModel**(*java_model=None*) [source]

Model fitted by LinearSVC.

New in version 2.2.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

property **coefficients**

Model coefficients of Linear SVM Classifier.

New in version 2.2.0.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

getAggregationDepth()
Gets the value of aggregationDepth or its default value.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getFitIntercept()
Gets the value of fitIntercept or its default value.

getLabelCol()
Gets the value of labelCol or its default value.

getMaxIter()
Gets the value of maxIter or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getRegParam()
Gets the value of regParam or its default value.

getStandardization()
Gets the value of standardization or its default value.

getThreshold()
Gets the value of threshold or its default value.

getTol()
Gets the value of tol or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

property **intercept**
Model intercept of Linear SVM Classifier.

New in version 2.2.0.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **numClasses**
Number of classes (values which the label can take).

New in version 2.1.0.

property **numFeatures**
Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)
Predict label for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setThreshold(*value*) [source]

Sets the value of **threshold**.

New in version 3.0.0.

standardization = *Param(parent='undefined', name='standardization', doc='whether to standardize the training features before fitting the model.')*

threshold = *Param(parent='undefined', name='threshold', doc='The threshold in binary classification applied to the linear model prediction. This threshold can be any real number, where Inf will make all predictions 0.0 and -Inf will make all predictions 1.0.')*

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

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

Logistic regression. This class supports multinomial logistic (softmax) and binomial logistic regression.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> bdf = sc.parallelize([
...     Row(label=1.0, weight=1.0, features=Vectors.dense(0.0, 5.0)),
...     Row(label=0.0, weight=2.0, features=Vectors.dense(1.0, 2.0)),
...     Row(label=1.0, weight=3.0, features=Vectors.dense(2.0, 1.0)),
...     Row(label=0.0, weight=4.0, features=Vectors.dense(3.0, 3.0))]).toDF()
>>> blor = LogisticRegression(weightCol="weight")
>>> blor.getRegParam()
0.0
>>> blor.setRegParam(0.01)
LogisticRegression...
>>> blor.getRegParam()
0.01
>>> blor.setMaxIter(10)
LogisticRegression...
>>> blor.getMaxIter()
10
>>> blor.clear(blor.maxIter)
>>> blorModel = blor.fit(bdf)
>>> blorModel.setFeaturesCol("features")
LogisticRegressionModel...
>>> blorModel.setProbabilityCol("newProbability")
LogisticRegressionModel...
>>> blorModel.getProbabilityCol()
'newProbability'
>>> blorModel.setThreshold(0.1)
LogisticRegressionModel...
>>> blorModel.getThreshold()
0.1
>>> blorModel.coefficients
DenseVector([-1.080..., -0.646...])
>>> blorModel.intercept
3.112...
>>> data_path = "data/mllib/sample_multiclass_classification_data.txt"
>>> mdf = spark.read.format("libsvm").load(data_path)
>>> mlor = LogisticRegression(regParam=0.1, elasticNetParam=1.0, family="multinomial")
>>> mlorModel = mlor.fit(mdf)
>>> mlorModel.coefficientMatrix
SparseMatrix(3, 4, [0, 1, 2, 3], [3, 2, 1], [1.87..., -2.75..., -0.50...], 1)
>>> mlorModel.interceptVector
DenseVector([0.04..., -0.42..., 0.37...])
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, 1.0))]).toDF()
>>> blorModel.predict(test0.head().features)
1.0
>>> result = blorModel.transform(test0).head()
>>> result.prediction
1.0
>>> result.newProbability
DenseVector([0.02..., 0.97...])
>>> result.rawPrediction
DenseVector([-3.54..., 3.54...])
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(2, [0], [1.0]))]).toDF()
>>> blorModel.transform(test1).head().prediction
1.0
>>> blor.setParams("vector")
Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
>>> lr_path = temp_path + "/lr"
>>> blor.save(lr_path)
>>> lr2 = LogisticRegression.load(lr_path)
>>> lr2.getRegParam()
0.01
>>> model_path = temp_path + "/lr_model"
>>> blorModel.save(model_path)
>>> model2 = LogisticRegressionModel.load(model_path)
>>> blorModel.coefficients[0] == model2.coefficients[0]
True
>>> blorModel.intercept == model2.intercept
True
>>> model2
LogisticRegressionModel: uid=..., numClasses=2, numFeatures=2
```

New in version 1.3.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

elasticNetParam = *Param(parent='undefined', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

family = *Param(parent='undefined', name='family', doc='The name of family which is a description of the label distribution to be used in the model. Supported options: auto, binomial, multinomial')*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**

- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getAggregationDepth()

Gets the value of aggregationDepth or its default value.

getElasticNetParam()

Gets the value of elasticNetParam or its default value.

getFamily()

Gets the value of **family** or its default value.

New in version 2.1.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getFitIntercept()

Gets the value of fitIntercept or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getLowerBoundsOnCoefficients()

Gets the value of **lowerBoundsOnCoefficients**

New in version 2.3.0.

getLowerBoundsOnIntercepts()

Gets the value of **lowerBoundsOnIntercepts**

New in version 2.3.0.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()

Gets the value of probabilityCol or its default value.

getRawPredictionCol()

Gets the value of rawPredictionCol or its default value.

getRegParam()

Gets the value of regParam or its default value.

getStandardization()

Gets the value of standardization or its default value.

getThreshold()

Get threshold for binary classification.

If **thresholds** is set with length 2 (i.e., binary classification), this returns the equivalent threshold: \(\frac{1}{1 + \frac{thresholds(0)}{thresholds(1)}}\). Otherwise, returns **threshold** if set or its default value if unset.

New in version 1.4.0.

getThresholds()

If **thresholds** is set, return its value. Otherwise, if **threshold** is set, return the equivalent thresholds for binary classification: (1-threshold, threshold). If neither are set, throw an error.

New in version 1.5.0.

getTol()

Gets the value of tol or its default value.

getUpperBoundsOnCoefficients()

Gets the value of **upperBoundsOnCoefficients**

New in version 2.3.0.

getUpperBoundsOnIntercepts()

Gets the value of **upperBoundsOnIntercepts**

New in version 2.3.0.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

lowerBoundsOnCoefficients = *Param(parent='undefined', name='lowerBoundsOnCoefficients', doc='The lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression.')*

lowerBoundsOnIntercepts = *Param(parent='undefined', name='lowerBoundsOnIntercepts', doc='The lower bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must beequal with 1 for binomial regression, or the number oflasses for multinomial regression.')*

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setAggregationDepth(*value*) [source]

Sets the value of **aggregationDepth**.

setElasticNetParam(*value*) [source]

Sets the value of **elasticNetParam**.

setFamily(*value*) [source]

Sets the value of **family**.

New in version 2.1.0.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setFitIntercept(*value*) [source]

Sets the value of **fitIntercept**.

setLabelCol(*value*)

Sets the value of **labelCol**.

New in version 3.0.0.

setLowerBoundsOnCoefficients(*value*) [source]

Sets the value of **lowerBoundsOnCoefficients**

New in version 2.3.0.

setLowerBoundsOnIntercepts(*value*) [source]

Sets the value of **lowerBoundsOnIntercepts**

New in version 2.3.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

setParams(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol='probability', rawPredictionCol='rawPrediction', standardization=True, weightCol=None, aggregationDepth=2, family='auto', lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lowerBoundsOnIntercepts=None, upperBoundsOnIntercepts=None*) [source]

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-6, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol="probability", rawPredictionCol="rawPrediction", standardization=True, weightCol=None, aggregationDepth=2, family="auto", lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lowerBoundsOnIntercepts=None, upperBoundsOnIntercepts=None): Sets params for logistic regression. If the threshold and thresholds Params are both set, they must be equivalent.

New in version 1.3.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)

Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setRegParam(*value*) [source]

Sets the value of **regParam**.

setStandardization(*value*) [source]

Sets the value of **standardization**.

setThreshold(*value*)

Sets the value of **threshold**. Clears value of **thresholds** if it has been set.

New in version 1.4.0.

setThresholds(*value*)

Sets the value of **thresholds**.

New in version 3.0.0.

setTol(*value*) [source]

Sets the value of **tol**.

setUpperBoundsOnCoefficients(*value*) [source]

Sets the value of **upperBoundsOnCoefficients**

New in version 2.3.0.

setUpperBoundsOnIntercepts(*value*) [source]

Sets the value of **upperBoundsOnIntercepts**

New in version 2.3.0.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

standardization = *Param(parent='undefined', name='standardization', doc='whether to standardize the training features before fitting the model.')*

threshold = *Param(parent='undefined', name='threshold', doc='Threshold in binary classification prediction, in range [0, 1]. If threshold and thresholds are both set, they must match.e.g. if threshold is p, then thresholds must be equal to [1-p, p].')*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

upperBoundsOnCoefficients = *Param(parent='undefined', name='upperBoundsOnCoefficients', doc='The upper bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression.')*

upperBoundsOnIntercepts = *Param(parent='undefined', name='upperBoundsOnIntercepts', doc='The upper bounds on intercepts if fitting under bound constrained optimization. The bound vector size must be equal with 1 for binomial regression, or the number of classes for multinomial regression.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**LogisticRegressionModel**(*java_model=None*) [source]

Model fitted by LogisticRegression.

New in version 1.3.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

property **coefficientMatrix**

Model coefficients.

New in version 2.1.0.

property **coefficients**

Model coefficients of binomial logistic regression. An exception is thrown in the case of multinomial logistic regression.

New in version 2.0.0.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

elasticNetParam = *Param(parent='undefined', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.')*

evaluate(*dataset*) [source]

Evaluates the model on a test dataset.

Parameters:
dataset – Test dataset to evaluate model on, where dataset is an instance of **pyspark.sql.DataFrame**

New in version 2.0.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

family = *Param(parent='undefined', name='family', doc='The name of family which is a description of the label distribution to be used in the model. Supported options: auto, binomial, multinomial')*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

getAggregationDepth()

Gets the value of aggregationDepth or its default value.

getElasticNetParam()

Gets the value of elasticNetParam or its default value.

getFamily()

Gets the value of **family** or its default value.

New in version 2.1.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getFitIntercept()

Gets the value of fitIntercept or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getLowerBoundsOnCoefficients()

Gets the value of **lowerBoundsOnCoefficients**

New in version 2.3.0.

getLowerBoundsOnIntercepts()

Gets the value of **lowerBoundsOnIntercepts**

New in version 2.3.0.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()

Gets the value of probabilityCol or its default value.

getRawPredictionCol()

Gets the value of rawPredictionCol or its default value.

getRegParam()

Gets the value of regParam or its default value.

getStandardization()

Gets the value of standardization or its default value.

getThreshold()

Get threshold for binary classification.

If **thresholds** is set with length 2 (i.e., binary classification), this returns the equivalent threshold: \(\frac{1}{1 + \frac{thresholds(0)}{thresholds(1)}}\). Otherwise, returns **threshold** if set or its default value if unset.

New in version 1.4.0.

getThresholds()

If **thresholds** is set, return its value. Otherwise, if **threshold** is set, return the equivalent thresholds for binary classification: (1-threshold, threshold). If neither are set, throw an error.

New in version 1.5.0.

getTol()

Gets the value of tol or its default value.

getUpperBoundsOnCoefficients()
Gets the value of **upperBoundsOnCoefficients**

New in version 2.3.0.

getUpperBoundsOnIntercepts()
Gets the value of **upperBoundsOnIntercepts**

New in version 2.3.0.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

property **hasSummary**
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

property **intercept**
Model intercept of binomial logistic regression. An exception is thrown in the case of multinomial logistic regression.

New in version 1.4.0.

property **interceptVector**
Model intercept.

New in version 2.1.0.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

lowerBoundsOnCoefficients = *Param(parent='undefined', name='lowerBoundsOnCoefficients', doc='The lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression.')*

lowerBoundsOnIntercepts = *Param(parent='undefined', name='lowerBoundsOnIntercepts', doc='The lower bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must beequal with 1 for binomial regression, or the number oflasses for multinomial regression.')*

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **numClasses**
Number of classes (values which the label can take).

New in version 2.1.0.

property **numFeatures**
Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)
Predict label for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)
Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setThreshold(*value*)

Sets the value of **threshold**. Clears value of **thresholds** if it has been set.

New in version 1.4.0.

setThresholds(*value*)

Sets the value of **thresholds**.

New in version 3.0.0.

standardization = *Param(parent='undefined', name='standardization', doc='whether to standardize the training features before fitting the model.')*

property **summary**

Gets summary (e.g. accuracy/precision/recall, objective history, total iterations) of model trained on the training set. An exception is thrown if *trainingSummary is None*.

New in version 2.0.0.

threshold = *Param(parent='undefined', name='threshold', doc='Threshold in binary classification prediction, in range [0, 1]. If threshold and thresholds are both set, they must match.e.g. if threshold is p, then thresholds must be equal to [1-p, p].')*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

upperBoundsOnCoefficients = *Param(parent='undefined', name='upperBoundsOnCoefficients', doc='The upper bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression.')*

upperBoundsOnIntercepts = *Param(parent='undefined', name='upperBoundsOnIntercepts', doc='The upper bounds on intercepts if fitting under bound constrained optimization. The bound vector size must be equal with 1 for binomial regression, or the number of classes for multinomial regression.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**LogisticRegressionSummary**(*java_obj=None*) [source]

Abstraction for Logistic Regression Results for a given model.

New in version 2.0.0.

property **accuracy**

Returns accuracy. (equals to the total number of correctly classified instances out of the total number of instances.)

New in version 2.3.0.

fMeasureByLabel(*beta=1.0*) [source]

Returns f-measure for each label (category).

New in version 2.3.0.

property **falsePositiveRateByLabel**

Returns false positive rate for each label (category).

New in version 2.3.0.

property **featuresCol**

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

property **labelCol**

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

property **labels**

Returns the sequence of labels in ascending order. This order matches the order used in metrics which are specified as arrays over labels, e.g., truePositiveRateByLabel.

Note: In most cases, it will be values {0.0, 1.0, …, numClasses-1}, However, if the training set is missing a label, then all of the arrays over labels (e.g., from truePositiveRateByLabel) will be of length numClasses-1 instead of the expected numClasses.

New in version 2.3.0.

property **precisionByLabel**

Returns precision for each label (category).

New in version 2.3.0.

property **predictionCol**

Field in “predictions” which gives the prediction of each class.

New in version 2.3.0.

property **predictions**

Dataframe outputted by the model’s *transform* method.

New in version 2.0.0.

property **probabilityCol**

Field in “predictions” which gives the probability of each class as a vector.

New in version 2.0.0.

property **recallByLabel**

Returns recall for each label (category).

New in version 2.3.0.

property **truePositiveRateByLabel**

Returns true positive rate for each label (category).

New in version 2.3.0.

weightedFMeasure(*beta=1.0*) [source]

Returns weighted averaged f-measure.

New in version 2.3.0.

property **weightedFalsePositiveRate**

Returns weighted false positive rate.

New in version 2.3.0.

property **weightedPrecision**

Returns weighted averaged precision.

New in version 2.3.0.

property **weightedRecall**

Returns weighted averaged recall. (equals to precision, recall and f-measure)

New in version 2.3.0.

property **weightedTruePositiveRate**

Returns weighted true positive rate. (equals to precision, recall and f-measure)

New in version 2.3.0.

class pyspark.ml.classification.**LogisticRegressionTrainingSummary**(*java_obj=None*) [source]

Abstraction for multinomial Logistic Regression Training results. Currently, the training summary ignores the training weights except for the objective trace.

New in version 2.0.0.

property **accuracy**

Returns accuracy. (equals to the total number of correctly classified instances out of the total number of instances.)

New in version 2.3.0.

fMeasureByLabel(*beta=1.0*)

Returns f-measure for each label (category).

New in version 2.3.0.

property **falsePositiveRateByLabel**

Returns false positive rate for each label (category).

New in version 2.3.0.

property **featuresCol**

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

property **labelCol**

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

property **labels**

Returns the sequence of labels in ascending order. This order matches the order used in metrics which are specified as arrays over labels, e.g., truePositiveRateByLabel.

Note: In most cases, it will be values {0.0, 1.0, …, numClasses-1}, However, if the training set is missing a label, then all of the arrays over labels (e.g., from truePositiveRateByLabel) will be of length numClasses-1 instead of the expected numClasses.

New in version 2.3.0.

property **objectiveHistory**

Objective function (scaled loss + regularization) at each iteration.

New in version 2.0.0.

property **precisionByLabel**

Returns precision for each label (category).

New in version 2.3.0.

property **predictionCol**

Field in “predictions” which gives the prediction of each class.

New in version 2.3.0.

property **predictions**

Dataframe outputted by the model’s *transform* method.

New in version 2.0.0.

property **probabilityCol**

Field in “predictions” which gives the probability of each class as a vector.

New in version 2.0.0.

property **recallByLabel**

Returns recall for each label (category).

New in version 2.3.0.

property **totalIterations**

Number of training iterations until termination.

New in version 2.0.0.

property **truePositiveRateByLabel**

Returns true positive rate for each label (category).

New in version 2.3.0.

weightedFMeasure(*beta=1.0*)

Returns weighted averaged f-measure.

New in version 2.3.0.

property **weightedFalsePositiveRate**

Returns weighted false positive rate.

New in version 2.3.0.

property **weightedPrecision**

Returns weighted averaged precision.

New in version 2.3.0.

property **weightedRecall**

Returns weighted averaged recall. (equals to precision, recall and f-measure)

New in version 2.3.0.

property **weightedTruePositiveRate**

Returns weighted true positive rate. (equals to precision, recall and f-measure)

New in version 2.3.0.

class pyspark.ml.classification.**BinaryLogisticRegressionSummary**(*java_obj=None*) [source]

Binary Logistic regression results for a given model.

New in version 2.0.0.

property **accuracy**

Returns accuracy. (equals to the total number of correctly classified instances out of the total number of instances.)

New in version 2.3.0.

property **areaUnderROC**

Computes the area under the receiver operating characteristic (ROC) curve.

> **Note:** This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

fMeasureByLabel(*beta=1.0*)

Returns f-measure for each label (category).

New in version 2.3.0.

property **fMeasureByThreshold**

Returns a dataframe with two fields (threshold, F-Measure) curve with beta = 1.0.

> **Note:** This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **falsePositiveRateByLabel**

Returns false positive rate for each label (category).

New in version 2.3.0.

property **featuresCol**

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

property **labelCol**

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

property **labels**

Returns the sequence of labels in ascending order. This order matches the order used in metrics which are specified as arrays over labels, e.g., truePositiveRateByLabel.

Note: In most cases, it will be values {0.0, 1.0, …, numClasses-1}, However, if the training set is missing a label, then all of the arrays over labels (e.g., from truePositiveRateByLabel) will be of length numClasses-1 instead of the expected numClasses.

New in version 2.3.0.

property **pr**

Returns the precision-recall curve, which is a Dataframe containing two fields recall, precision with (0.0, 1.0) prepended to it.

Note: This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **precisionByLabel**

Returns precision for each label (category).

New in version 2.3.0.

property **precisionByThreshold**

Returns a dataframe with two fields (threshold, precision) curve. Every possible probability obtained in transforming the dataset are used as thresholds used in calculating the precision.

Note: This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **predictionCol**

Field in "predictions" which gives the prediction of each class.

New in version 2.3.0.

property **predictions**

Dataframe outputted by the model's *transform* method.

New in version 2.0.0.

property **probabilityCol**

Field in "predictions" which gives the probability of each class as a vector.

New in version 2.0.0.

property **recallByLabel**

Returns recall for each label (category).

New in version 2.3.0.

property **recallByThreshold**

Returns a dataframe with two fields (threshold, recall) curve. Every possible probability obtained in transforming the dataset are used as thresholds used in calculating the recall.

Note: This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **roc**

Returns the receiver operating characteristic (ROC) curve, which is a Dataframe having two fields (FPR, TPR) with (0.0, 0.0) prepended and (1.0, 1.0) appended to it.

See also: Wikipedia reference

Note: This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **truePositiveRateByLabel**

Returns true positive rate for each label (category).

New in version 2.3.0.

weightedFMeasure(*beta=1.0*)

Returns weighted averaged f-measure.

New in version 2.3.0.

property **weightedFalsePositiveRate**

Returns weighted false positive rate.

New in version 2.3.0.

property **weightedPrecision**

Returns weighted averaged precision.

New in version 2.3.0.

property **weightedRecall**

Returns weighted averaged recall. (equals to precision, recall and f-measure)

New in version 2.3.0.

property **weightedTruePositiveRate**

Returns weighted true positive rate. (equals to precision, recall and f-measure)

New in version 2.3.0.

class pyspark.ml.classification.**BinaryLogisticRegressionTrainingSummary**(*java_obj=None*) [source]

Binary Logistic regression training results for a given model.

New in version 2.0.0.

property **accuracy**

Returns accuracy. (equals to the total number of correctly classified instances out of the total number of instances.)

New in version 2.3.0.

property **areaUnderROC**

Computes the area under the receiver operating characteristic (ROC) curve.

Note: This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

fMeasureByLabel(*beta=1.0*)

Returns f-measure for each label (category).

New in version 2.3.0.

property **fMeasureByThreshold**

Returns a dataframe with two fields (threshold, F-Measure) curve with beta = 1.0.

> **Note:** This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **falsePositiveRateByLabel**

Returns false positive rate for each label (category).

New in version 2.3.0.

property **featuresCol**

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

property **labelCol**

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

property **labels**

Returns the sequence of labels in ascending order. This order matches the order used in metrics which are specified as arrays over labels, e.g., truePositiveRateByLabel.

Note: In most cases, it will be values {0.0, 1.0, …, numClasses-1}, However, if the training set is missing a label, then all of the arrays over labels (e.g., from truePositiveRateByLabel) will be of length numClasses-1 instead of the expected numClasses.

New in version 2.3.0.

property **objectiveHistory**

Objective function (scaled loss + regularization) at each iteration.

New in version 2.0.0.

property **pr**

Returns the precision-recall curve, which is a Dataframe containing two fields recall, precision with (0.0, 1.0) prepended to it.

> **Note:** This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **precisionByLabel**

Returns precision for each label (category).

New in version 2.3.0.

property **precisionByThreshold**

Returns a dataframe with two fields (threshold, precision) curve. Every possible probability obtained in transforming the dataset are used as thresholds used in calculating the precision.

> **Note:** This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **predictionCol**

Field in “predictions” which gives the prediction of each class.

New in version 2.3.0.

property **predictions**

Dataframe outputted by the model’s *transform* method.

New in version 2.0.0.

property **probabilityCol**

Field in “predictions” which gives the probability of each class as a vector.

New in version 2.0.0.

property **recallByLabel**

Returns recall for each label (category).

New in version 2.3.0.

property **recallByThreshold**

Returns a dataframe with two fields (threshold, recall) curve. Every possible probability obtained in transforming the dataset are used as thresholds used in calculating the recall.

> **Note:** This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **roc**

Returns the receiver operating characteristic (ROC) curve, which is a Dataframe having two fields (FPR, TPR) with (0.0, 0.0) prepended and (1.0, 1.0) appended to it.

> **See also:** Wikipedia reference

> **Note:** This ignores instance weights (setting all to 1.0) from *LogisticRegression.weightCol*. This will change in later

Spark versions.

New in version 2.0.0.

property **totalIterations**

Number of training iterations until termination.

New in version 2.0.0.

property **truePositiveRateByLabel**

Returns true positive rate for each label (category).

New in version 2.3.0.

weightedFMeasure(*beta=1.0*)

Returns weighted averaged f-measure.

New in version 2.3.0.

property **weightedFalsePositiveRate**

Returns weighted false positive rate.

New in version 2.3.0.

property **weightedPrecision**

Returns weighted averaged precision.

New in version 2.3.0.

property **weightedRecall**

Returns weighted averaged recall. (equals to precision, recall and f-measure)

New in version 2.3.0.

property **weightedTruePositiveRate**

Returns weighted true positive rate. (equals to precision, recall and f-measure)

New in version 2.3.0.

class pyspark.ml.classification.**DecisionTreeClassifier**(*featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', seed=None, weightCol=None, leafCol='', minWeightFractionPerNode=0.0*) [source]

Decision tree learning algorithm for classification. It supports both binary and multiclass labels, as well as both continuous and categorical features.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> dt = DecisionTreeClassifier(maxDepth=2, labelCol="indexed", leafCol="leafId")
>>> model = dt.fit(td)
>>> model.getLabelCol()
'indexed'
>>> model.setFeaturesCol("features")
DecisionTreeClassificationModel...
>>> model.numNodes
3
>>> model.depth
1
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> model.numClasses
2
>>> print(model.toDebugString)
DecisionTreeClassificationModel...depth=1, numNodes=3...
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.predict(test0.head().features)
0.0
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> result.probability
DenseVector([1.0, 0.0])
>>> result.rawPrediction
DenseVector([1.0, 0.0])
>>> result.leafId
0.0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
>>> dtc_path = temp_path + "/dtc"
>>> dt.save(dtc_path)
>>> dt2 = DecisionTreeClassifier.load(dtc_path)
>>> dt2.getMaxDepth()
2
>>> model_path = temp_path + "/dtc_model"
>>> model.save(model_path)
>>> model2 = DecisionTreeClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

```
>>> df3 = spark.createDataFrame([
...     (1.0, 0.2, Vectors.dense(1.0)),
...     (1.0, 0.8, Vectors.dense(1.0)),
...     (0.0, 1.0, Vectors.sparse(1, [], []))], ["label", "weight", "features"])
>>> si3 = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model3 = si3.fit(df3)
>>> td3 = si_model3.transform(df3)
>>> dt3 = DecisionTreeClassifier(maxDepth=2, weightCol="weight", labelCol="indexed")
>>> model3 = dt3.fit(td3)
>>> print(model3.toDebugString)
DecisionTreeClassificationModel...depth=1, numNodes=3...
```

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.6.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()

Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()

Gets the value of minWeightFractionPerNode or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()
Gets the value of probabilityCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getSeed()
Gets the value of seed or its default value.

getThresholds()
Gets the value of thresholds or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: entropy, gini')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setCacheNodeIds(*value*) [source]
Sets the value of **cacheNodeIds**.

setCheckpointInterval(*value*) [source]
Sets the value of **checkpointInterval**.

New in version 1.4.0.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setImpurity(*value*) [source]
Sets the value of **impurity**.

New in version 1.4.0.

setLabelCol(*value*)
Sets the value of **labelCol**.

New in version 3.0.0.

setLeafCol(*value*)
Sets the value of **leafCol**.

setMaxBins(*value*) [source]
Sets the value of **maxBins**.

setMaxDepth(*value*) [source]
Sets the value of **maxDepth**.

setMaxMemoryInMB(*value*) [source]
Sets the value of **maxMemoryInMB**.

setMinInfoGain(*value*) [source]
Sets the value of **minInfoGain**.

setMinInstancesPerNode(*value*) [source]
Sets the value of **minInstancesPerNode**.

setMinWeightFractionPerNode(*value*) [source]
Sets the value of **minWeightFractionPerNode**.

New in version 3.0.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity="gini"*, *seed=None*, *weightCol=None*, *leafCol=""*, *minWeightFractionPerNode=0.0*) [source]
Sets params for the DecisionTreeClassifier.

New in version 1.4.0.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)
Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)
Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setSeed(*value*) [source]
Sets the value of **seed**.

New in version 1.4.0.

setThresholds(*value*)
Sets the value of **thresholds**.

New in version 3.0.0.

setWeightCol(*value*) [source]
Sets the value of **weightCol**.

New in version 3.0.0.

supportedImpurities = *['entropy', 'gini']*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**DecisionTreeClassificationModel**(*java_model=None*) [source]
Model fitted by DecisionTreeClassifier.

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

property **depth**
Return depth of the decision tree.

New in version 1.5.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

property **featureImportances**

Estimate of the importance of each feature.

This generalizes the idea of “Gini” importance to other losses, following the explanation of Gini importance from “Random Forests” documentation by Leo Breiman and Adele Cutler, and following the implementation from scikit-learn.

This feature importance is calculated as follows:

- importance(feature j) = sum (over nodes which split on feature j) of the gain, where gain is scaled by the number of instances passing through node
- Normalize importances for tree to sum to 1.

Note: Feature importance for single decision trees can have high variance due to correlated predictor variables. Consider using a **RandomForestClassifier** to determine feature importance instead.

New in version 2.0.0.

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.6.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()

Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()

Gets the value of minWeightFractionPerNode or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()

Gets the value of probabilityCol or its default value.

getRawPredictionCol()

Gets the value of rawPredictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getThresholds()

Gets the value of thresholds or its default value.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: entropy, gini')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **numClasses**

Number of classes (values which the label can take).

New in version 2.1.0.

property **numFeatures**

Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **numNodes**

Return number of nodes of the decision tree.

New in version 1.5.0.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)

Predict label for the given features.

New in version 3.0.0.

predictLeaf(*value*)

Predict the indices of the leaves corresponding to the feature vector.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setLeafCol(*value*)

Sets the value of **leafCol**.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)

Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setThresholds(*value*)

Sets the value of **thresholds**.

New in version 3.0.0.

supportedImpurities = *['entropy', 'gini']*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

property **toDebugString**

Full description of model.

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**GBTClassifier**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *lossType='logistic'*, *maxIter=20*, *stepSize=0.1*, *seed=None*, *subsamplingRate=1.0*, *impurity='variance'*, *featureSubsetStrategy='all'*, *validationTol=0.01*, *validationIndicatorCol=None*, *leafCol=''*, *minWeightFractionPerNode=0.0*, *weightCol=None*) [source]

Gradient-Boosted Trees (GBTs) learning algorithm for classification. It supports binary labels, as well as both continuous and categorical features.

The implementation is based upon: J.H. Friedman. “Stochastic Gradient Boosting.” 1999.

Notes on Gradient Boosting vs. TreeBoost: - This implementation is for Stochastic Gradient Boosting, not for TreeBoost. - Both algorithms learn tree ensembles by minimizing loss functions. - TreeBoost (Friedman, 1999) additionally modifies the outputs at tree leaf nodes based on the loss function, whereas the original gradient boosting method does not. - We expect to implement TreeBoost in the future: SPARK-4240

Note:	Multiclass labels are not currently supported.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> gbt = GBTClassifier(maxIter=5, maxDepth=2, labelCol="indexed", seed=42,
...     leafCol="leafId")
>>> gbt.setMaxIter(5)
GBTClassifier...
>>> gbt.setMinWeightFractionPerNode(0.049)
GBTClassifier...
>>> gbt.getMaxIter()
5
>>> gbt.getFeatureSubsetStrategy()
'all'
>>> model = gbt.fit(td)
>>> model.getLabelCol()
'indexed'
>>> model.setFeaturesCol("features")
GBTClassificationModel...
>>> model.setThresholds([0.3, 0.7])
GBTClassificationModel...
>>> model.getThresholds()
[0.3, 0.7]
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 0.1, 0.1, 0.1, 0.1])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.predict(test0.head().features)
0.0
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> result.leafId
DenseVector([0.0, 0.0, 0.0, 0.0, 0.0])
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
>>> model.totalNumNodes
15
>>> print(model.toDebugString)
GBTClassificationModel...numTrees=5...
>>> gbtc_path = temp_path + "gbtc"
>>> gbt.save(gbtc_path)
>>> gbt2 = GBTClassifier.load(gbtc_path)
>>> gbt2.getMaxDepth()
2
>>> model_path = temp_path + "gbtc_model"
>>> model.save(model_path)
>>> model2 = GBTClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
>>> model.treeWeights == model2.treeWeights
True
>>> model.trees
[DecisionTreeRegressionModel...depth=..., DecisionTreeRegressionModel...]
>>> validation = spark.createDataFrame([(0.0, Vectors.dense(-1.0),)],
...              ["indexed", "features"])
>>> model.evaluateEachIteration(validation)
[0.25..., 0.23..., 0.21..., 0.19..., 0.18...]
>>> model.numClasses
2
>>> gbt = gbt.setValidationIndicatorCol("validationIndicator")
>>> gbt.getValidationIndicatorCol()
'validationIndicator'
>>> gbt.getValidationTol()
0.01
```

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featureSubsetStrategy = *Param(parent='undefined', name='featureSubsetStrategy', doc="The number of features to consider for splits at each tree node. Supported options: 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(number of features)), 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features). default = 'auto'")*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of `pyspark.sql.DataFrame`
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of `pyspark.sql.DataFrame`.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeatureSubsetStrategy()

Gets the value of featureSubsetStrategy or its default value.

New in version 1.4.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.4.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getLossType()

Gets the value of lossType or its default value.

New in version 1.4.0.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxIter()

Gets the value of maxIter or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()

Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()

Gets the value of minWeightFractionPerNode or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()

Gets the value of probabilityCol or its default value.

getRawPredictionCol()

Gets the value of rawPredictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getStepSize()

Gets the value of stepSize or its default value.

getSubsamplingRate()

Gets the value of subsamplingRate or its default value.

New in version 1.4.0.

getThresholds()

Gets the value of thresholds or its default value.

getValidationIndicatorCol()
Gets the value of validationIndicatorCol or its default value.

getValidationTol()
Gets the value of validationTol or its default value.

New in version 3.0.0.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: variance')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

lossType = *Param(parent='undefined', name='lossType', doc='Loss function which GBT tries to minimize (case-insensitive). Supported options: logistic')*

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setCacheNodeIds(*value*) [source]
Sets the value of **cacheNodeIds**.

setCheckpointInterval(*value*) [source]
Sets the value of **checkpointInterval**.

New in version 1.4.0.

setFeatureSubsetStrategy(*value*) [source]
Sets the value of **featureSubsetStrategy**.

New in version 2.4.0.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setImpurity(*value*) [source]

Sets the value of **impurity**.

New in version 1.4.0.

setLabelCol(*value*)
Sets the value of **labelCol**.

New in version 3.0.0.

setLeafCol(*value*)
Sets the value of **leafCol**.

setLossType(*value*) [source]
Sets the value of **lossType**.

New in version 1.4.0.

setMaxBins(*value*) [source]
Sets the value of **maxBins**.

setMaxDepth(*value*) [source]
Sets the value of **maxDepth**.

setMaxIter(*value*) [source]
Sets the value of **maxIter**.

New in version 1.4.0.

setMaxMemoryInMB(*value*) [source]
Sets the value of **maxMemoryInMB**.

setMinInfoGain(*value*) [source]
Sets the value of **minInfoGain**.

setMinInstancesPerNode(*value*) [source]
Sets the value of **minInstancesPerNode**.

setMinWeightFractionPerNode(*value*) [source]
Sets the value of **minWeightFractionPerNode**.

New in version 3.0.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *lossType="logistic"*, *maxIter=20*, *stepSize=0.1*, *seed=None*, *subsamplingRate=1.0*, *impurity="variance"*, *featureSubsetStrategy="all"*, *validationTol=0.01*, *validationIndicatorCol=None*, *leafCol=""*, *minWeightFractionPerNode=0.0*, *weightCol=None*) [source]
Sets params for Gradient Boosted Tree Classification.

New in version 1.4.0.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)
Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)
Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setSeed(*value*) [source]
Sets the value of **seed**.

New in version 1.4.0.

setStepSize(*value*) [source]
Sets the value of **stepSize**.

New in version 1.4.0.

setSubsamplingRate(*value*) [source]
Sets the value of **subsamplingRate**.

New in version 1.4.0.

setThresholds(*value*)
Sets the value of **thresholds**.

New in version 3.0.0.

setValidationIndicatorCol(*value*) [source]
Sets the value of **validationIndicatorCol**.

New in version 3.0.0.

setWeightCol(*value*) [source]
Sets the value of **weightCol**.

New in version 3.0.0.

stepSize = *Param(parent='undefined', name='stepSize', doc='Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.')*

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedFeatureSubsetStrategies = *['auto', 'all', 'onethird', 'sqrt', 'log2']*

supportedImpurities = *['variance']*

supportedLossTypes = *['logistic']*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

validationIndicatorCol = *Param(parent='undefined', name='validationIndicatorCol', doc='name of the column that indicates whether each row is for training or for validation. False indicates training; true indicates validation.')*

validationTol = *Param(parent='undefined', name='validationTol', doc='Threshold for stopping early when fit with validation is used. If the error rate on the validation input changes by less than the validationTol, then learning will stop early (before `maxIter`). This parameter is ignored when fit without validation is used.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**GBTClassificationModel**(*java_model=None*) [source]

Model fitted by GBTClassifier.

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

evaluateEachIteration(*dataset*) [source]

Method to compute error or loss for every iteration of gradient boosting.

Parameters:
dataset – Test dataset to evaluate model on, where dataset is an instance of **pyspark.sql.DataFrame**

New in version 2.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

property **featureImportances**

Estimate of the importance of each feature.

Each feature's importance is the average of its importance across all trees in the ensemble The importance vector is normalized to sum to 1. This method is suggested by Hastie et al. (Hastie, Tibshirani, Friedman. "The Elements of Statistical Learning, 2nd Edition." 2001.) and follows the implementation from scikit-learn.

See also: **DecisionTreeClassificationModel.featureImportances**

New in version 2.0.0.

featureSubsetStrategy = *Param(parent='undefined', name='featureSubsetStrategy', doc="The number of features to consider for splits at each tree node. Supported options: 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(number of features)), 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features). default = 'auto'")*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeatureSubsetStrategy()

Gets the value of featureSubsetStrategy or its default value.

New in version 1.4.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.4.0.

getLabelCol()
Gets the value of labelCol or its default value.

getLeafCol()
Gets the value of leafCol or its default value.

getLossType()
Gets the value of lossType or its default value.

New in version 1.4.0.

getMaxBins()
Gets the value of maxBins or its default value.

getMaxDepth()
Gets the value of maxDepth or its default value.

getMaxIter()
Gets the value of maxIter or its default value.

getMaxMemoryInMB()
Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()
Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()
Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()
Gets the value of minWeightFractionPerNode or its default value.

property **getNumTrees**
Number of trees in ensemble.

New in version 2.0.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getProbabilityCol()
Gets the value of probabilityCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getSeed()
Gets the value of seed or its default value.

getStepSize()
Gets the value of stepSize or its default value.

getSubsamplingRate()
Gets the value of subsamplingRate or its default value.

New in version 1.4.0.

getThresholds()
Gets the value of thresholds or its default value.

getValidationIndicatorCol()
Gets the value of validationIndicatorCol or its default value.

getValidationTol()
Gets the value of validationTol or its default value.

New in version 3.0.0.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: variance')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

lossType = *Param(parent='undefined', name='lossType', doc='Loss function which GBT tries to minimize (case-insensitive). Supported options: logistic')*

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **numClasses**

Number of classes (values which the label can take).

New in version 2.1.0.

property **numFeatures**

Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)

Predict label for the given features.

New in version 3.0.0.

predictLeaf(*value*)

Predict the indices of the leaves corresponding to the feature vector.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setLeafCol(*value*)

Sets the value of **leafCol**.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)

Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setThresholds(*value*)

Sets the value of **thresholds**.

New in version 3.0.0.

stepSize = *Param(parent='undefined', name='stepSize', doc='Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.')*

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedFeatureSubsetStrategies = *['auto', 'all', 'onethird', 'sqrt', 'log2']*

supportedImpurities = *['variance']*

supportedLossTypes = *['logistic']*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

property **toDebugString**

Full description of model.

New in version 2.0.0.

property **totalNumNodes**

Total number of nodes, summed over all trees in the ensemble.

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

property **treeWeights**

Return the weights for each tree

New in version 1.5.0.

property **trees**

Trees in this ensemble. Warning: These have null parent Estimators.

New in version 2.0.0.

validationIndicatorCol = *Param(parent='undefined', name='validationIndicatorCol', doc='name of the column that indicates whether each row is for training or for validation. False indicates training; true indicates validation.')*

validationTol = *Param(parent='undefined', name='validationTol', doc='Threshold for stopping early when fit with validation is used. If the error rate on the validation input changes by less than the validationTol, then learning will stop early (before `maxIter`). This parameter is ignored when fit without validation is used.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**RandomForestClassifier**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *probabilityCol='probability'*, *rawPredictionCol='rawPrediction'*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity='gini'*, *numTrees=20*, *featureSubsetStrategy='auto'*, *seed=None*, *subsamplingRate=1.0*, *leafCol=''*, *minWeightFractionPerNode=0.0*)

[source]

Random Forest learning algorithm for classification. It supports both binary and multiclass labels, as well as both continuous and categorical features.

```
>>> import numpy
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> si_model = stringIndexer.fit(df)
>>> td = si_model.transform(df)
>>> rf = RandomForestClassifier(numTrees=3, maxDepth=2, labelCol="indexed", seed=42,
...     leafCol="leafId")
>>> model = rf.fit(td)
>>> model.getLabelCol()
'indexed'
>>> model.setFeaturesCol("features")
RandomForestClassificationModel...
>>> model.setRawPredictionCol("newRawPrediction")
RandomForestClassificationModel...
>>> model.getRawPredictionCol()
'newRawPrediction'
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 1.0, 1.0])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.predict(test0.head().features)
0.0
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> numpy.argmax(result.probability)
0
>>> numpy.argmax(result.newRawPrediction)
0
>>> result.leafId
DenseVector([0.0, 0.0, 0.0])
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
>>> model.trees
[DecisionTreeClassificationModel...depth=..., DecisionTreeClassificationModel...]
>>> rfc_path = temp_path + "/rfc"
>>> rf.save(rfc_path)
>>> rf2 = RandomForestClassifier.load(rfc_path)
>>> rf2.getNumTrees()
3
>>> model_path = temp_path + "/rfc_model"
>>> model.save(model_path)
>>> model2 = RandomForestClassificationModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or*

disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featureSubsetStrategy = *Param(parent='undefined', name='featureSubsetStrategy', doc="The number of features to consider for splits at each tree node. Supported options: 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(number of features)), 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features). default = 'auto'")*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeatureSubsetStrategy()

Gets the value of featureSubsetStrategy or its default value.

New in version 1.4.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.6.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()
Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()
Gets the value of minWeightFractionPerNode or its default value.

getNumTrees()
Gets the value of numTrees or its default value.

New in version 1.4.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getProbabilityCol()
Gets the value of probabilityCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getSeed()
Gets the value of seed or its default value.

getSubsamplingRate()
Gets the value of subsamplingRate or its default value.

New in version 1.4.0.

getThresholds()
Gets the value of thresholds or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: entropy, gini')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

numTrees = *Param(parent='undefined', name='numTrees', doc='Number of trees to train (>= 1).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setCacheNodeIds(*value*) [source]
Sets the value of **cacheNodeIds**.

setCheckpointInterval(*value*) [source]
Sets the value of **checkpointInterval**.

setFeatureSubsetStrategy(*value*) [source]
Sets the value of **featureSubsetStrategy**.

New in version 2.4.0.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setImpurity(*value*) [source]
Sets the value of **impurity**.

New in version 1.4.0.

setLabelCol(*value*)
Sets the value of **labelCol**.

New in version 3.0.0.

setLeafCol(*value*)
Sets the value of **leafCol**.

setMaxBins(*value*) [source]
Sets the value of **maxBins**.

setMaxDepth(*value*) [source]
Sets the value of **maxDepth**.

setMaxMemoryInMB(*value*) [source]
Sets the value of **maxMemoryInMB**.

setMinInfoGain(*value*) [source]
Sets the value of **minInfoGain**.

setMinInstancesPerNode(*value*) [source]
Sets the value of **minInstancesPerNode**.

setNumTrees(*value*) [source]
Sets the value of **numTrees**.

New in version 1.4.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *seed=None*, *impurity="gini"*, *numTrees=20*, *featureSubsetStrategy="auto"*, *subsamplingRate=1.0*, *leafCol=""*, *minWeightFractionPerNode=0.0*) [source]
Sets params for linear classification.

New in version 1.4.0.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)
Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)
Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setSeed(*value*) [source]
Sets the value of **seed**.

setSubsamplingRate(*value*) [source]
Sets the value of **subsamplingRate**.

New in version 1.4.0.

setThresholds(*value*)
Sets the value of **thresholds**.

New in version 3.0.0.

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedFeatureSubsetStrategies = *['auto', 'all', 'onethird', 'sqrt', 'log2']*

supportedImpurities = *['entropy', 'gini']*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**RandomForestClassificationModel**(*java_model=None*) [source]

Model fitted by RandomForestClassifier.

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

property **featureImportances**

Estimate of the importance of each feature.

Each feature's importance is the average of its importance across all trees in the ensemble The importance vector is normalized to sum to 1. This method is suggested by Hastie et al. (Hastie, Tibshirani, Friedman. "The Elements of Statistical Learning, 2nd Edition." 2001.) and follows the implementation from scikit-learn.

See also: **DecisionTreeClassificationModel.featureImportances**

New in version 2.0.0.

featureSubsetStrategy = *Param(parent='undefined', name='featureSubsetStrategy', doc="The number of features to consider for splits at each tree node. Supported options: 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(number of features)), 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features). default = 'auto'")*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeatureSubsetStrategy()

Gets the value of featureSubsetStrategy or its default value.

New in version 1.4.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.6.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()

Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()

Gets the value of minWeightFractionPerNode or its default value.

property **getNumTrees**

Number of trees in ensemble.

New in version 2.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()

Gets the value of probabilityCol or its default value.

getRawPredictionCol()

Gets the value of rawPredictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getSubsamplingRate()

Gets the value of subsamplingRate or its default value.

New in version 1.4.0.

getThresholds()

Gets the value of thresholds or its default value.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: entropy, gini')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **numClasses**

Number of classes (values which the label can take).

New in version 2.1.0.

property **numFeatures**

Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

numTrees = *Param(parent='undefined', name='numTrees', doc='Number of trees to train (>= 1).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)

Predict label for the given features.

New in version 3.0.0.

predictLeaf(*value*)

Predict the indices of the leaves corresponding to the feature vector.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setLeafCol(*value*)
Sets the value of **leafCol**.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)
Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)
Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setThresholds(*value*)
Sets the value of **thresholds**.

New in version 3.0.0.

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedFeatureSubsetStrategies = *['auto', 'all', 'onethird', 'sqrt', 'log2']*

supportedImpurities = *['entropy', 'gini']*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

property **toDebugString**
Full description of model.

New in version 2.0.0.

property **totalNumNodes**
Total number of nodes, summed over all trees in the ensemble.

New in version 2.0.0.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

property **treeWeights**
Return the weights for each tree

New in version 1.5.0.

property **trees**
Trees in this ensemble. Warning: These have null parent Estimators.

New in version 2.0.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**NaiveBayes**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *probabilityCol='probability'*, *rawPredictionCol='rawPrediction'*, *smoothing=1.0*, *modelType='multinomial'*, *thresholds=None*, *weightCol=None*) [source]

Naive Bayes Classifiers. It supports both Multinomial and Bernoulli NB. Multinomial NB can handle finitely supported discrete data. For example, by converting documents into TF-IDF vectors, it can be used for document classification. By making every vector a binary (0/1) data, it can also be used as Bernoulli NB. The input feature values for Multinomial NB and Bernoulli NB must be nonnegative. Since 3.0.0, it supports Complement NB which is an adaptation of the Multinomial NB.

Specifically, Complement NB uses statistics from the complement of each class to compute the model's coefficients. The inventors of Complement NB show empirically that the parameter estimates for CNB are more stable than those for Multinomial NB. Like Multinomial NB, the input feature values for Complement NB must be nonnegative. Since 3.0.0, it also supports Gaussian NB <://en.wikipedia.org/wiki/Naive_Bayes_classifier#Gaussian_naive_Bayes>`_. which can handle continuous data.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     Row(label=0.0, weight=0.1, features=Vectors.dense([0.0, 0.0])),
...     Row(label=0.0, weight=0.5, features=Vectors.dense([0.0, 1.0])),
...     Row(label=1.0, weight=1.0, features=Vectors.dense([1.0, 0.0]))])
>>> nb = NaiveBayes(smoothing=1.0, modelType="multinomial", weightCol="weight")
>>> model = nb.fit(df)
>>> model.setFeaturesCol("features")
NaiveBayesModel...
>>> model.getSmoothing()
1.0
>>> model.pi
DenseVector([-0.81..., -0.58...])
>>> model.theta
DenseMatrix(2, 2, [-0.91..., -0.51..., -0.40..., -1.09...], 1)
>>> model.sigma
DenseMatrix(0, 0, [...], ...)
>>> test0 = sc.parallelize([Row(features=Vectors.dense([1.0, 0.0]))]).toDF()
>>> model.predict(test0.head().features)
1.0
>>> result = model.transform(test0).head()
>>> result.prediction
1.0
>>> result.probability
DenseVector([0.32..., 0.67...])
>>> result.rawPrediction
DenseVector([-1.72..., -0.99...])
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(2, [0], [1.0]))]).toDF()
>>> model.transform(test1).head().prediction
1.0
>>> nb_path = temp_path + "/nb"
>>> nb.save(nb_path)
>>> nb2 = NaiveBayes.load(nb_path)
>>> nb2.getSmoothing()
1.0
>>> model_path = temp_path + "/nb_model"
>>> model.save(model_path)
>>> model2 = NaiveBayesModel.load(model_path)
>>> model.pi == model2.pi
True
>>> model.theta == model2.theta
True
>>> nb = nb.setThresholds([0.01, 10.00])
>>> model3 = nb.fit(df)
>>> result = model3.transform(test0).head()
>>> result.prediction
0.0
>>> nb3 = NaiveBayes().setModelType("gaussian")
>>> model4 = nb3.fit(df)
>>> model4.getModelType()
'gaussian'
>>> model4.sigma
DenseMatrix(2, 2, [0.0, 0.25, 0.0, 0.0], 1)
>>> nb5 = NaiveBayes(smoothing=1.0, modelType="complement", weightCol="weight")
>>> model5 = nb5.fit(df)
>>> model5.getModelType()
'complement'
>>> model5.theta
DenseMatrix(2, 2, [...], 1)
>>> model5.sigma
DenseMatrix(0, 0, [...], ...)
```

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.

- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getLabelCol()
Gets the value of labelCol or its default value.

getModelType()
Gets the value of modelType or its default value.

New in version 1.5.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getProbabilityCol()
Gets the value of probabilityCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getSmoothing()
Gets the value of smoothing or its default value.

New in version 1.5.0.

getThresholds()
Gets the value of thresholds or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

modelType = *Param(parent='undefined', name='modelType', doc='The model type which is a string (case-sensitive). Supported options: multinomial (default), bernoulli and gaussian.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setLabelCol(*value*)
Sets the value of **labelCol**.

New in version 3.0.0.

setModelType(*value*) [source]
Sets the value of **modelType**.

New in version 1.5.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*,

rawPredictionCol="rawPrediction", smoothing=1.0, modelType="multinomial", thresholds=None, weightCol=None) [source]

Sets params for Naive Bayes.

New in version 1.5.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)

Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setSmoothing(*value*) [source]

Sets the value of **smoothing**.

New in version 1.5.0.

setThresholds(*value*)

Sets the value of **thresholds**.

New in version 3.0.0.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

smoothing = *Param(parent='undefined', name='smoothing', doc='The smoothing parameter, should be >= 0, default is 1.0')*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**NaiveBayesModel**(*java_model=None*) [source]

Model fitted by NaiveBayes.

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getFeaturesCol()

Gets the value of featuresCol or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getModelType()

Gets the value of modelType or its default value.

New in version 1.5.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()

Gets the value of probabilityCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getSmoothing()
Gets the value of smoothing or its default value.

New in version 1.5.0.

getThresholds()
Gets the value of thresholds or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

modelType = *Param(parent='undefined', name='modelType', doc='The model type which is a string (case-sensitive). Supported options: multinomial (default), bernoulli and gaussian.')*

property **numClasses**
Number of classes (values which the label can take).

New in version 2.1.0.

property **numFeatures**
Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

property **pi**
log of class priors.

New in version 2.0.0.

predict(*value*)
Predict label for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)
Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)
Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setThresholds(*value*)
Sets the value of **thresholds**.

New in version 3.0.0.

property **sigma**
variance of each feature.

New in version 3.0.0.

smoothing = *Param(parent='undefined', name='smoothing', doc='The smoothing parameter, should be >= 0, default is 1.0')*

property **theta**

log of class conditional probabilities.

New in version 2.0.0.

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**MultilayerPerceptronClassifier**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxIter=100*, *tol=1e-06*, *seed=None*, *layers=None*, *blockSize=128*, *stepSize=0.03*, *solver='l-bfgs'*, *initialWeights=None*, *probabilityCol='probability'*, *rawPredictionCol='rawPrediction'*) [source]

Classifier trainer based on the Multilayer Perceptron. Each layer has sigmoid activation function, output layer has softmax. Number of inputs has to be equal to the size of feature vectors. Number of outputs has to be equal to the total number of labels.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (0.0, Vectors.dense([0.0, 0.0])),
...     (1.0, Vectors.dense([0.0, 1.0])),
...     (1.0, Vectors.dense([1.0, 0.0])),
...     (0.0, Vectors.dense([1.0, 1.0]))], ["label", "features"])
>>> mlp = MultilayerPerceptronClassifier(layers=[2, 2, 2], blockSize=1, seed=123)
>>> mlp.setMaxIter(100)
MultilayerPerceptronClassifier...
>>> mlp.getMaxIter()
100
>>> model = mlp.fit(df)
>>> model.setFeaturesCol("features")
MultilayerPerceptronClassificationModel...
>>> model.layers
[2, 2, 2]
>>> model.weights.size
12
>>> testDF = spark.createDataFrame([
...     (Vectors.dense([1.0, 0.0]),),
...     (Vectors.dense([0.0, 0.0]),)], ["features"])
>>> model.predict(testDF.head().features)
1.0
>>> model.transform(testDF).select("features", "prediction").show()
+---------+----------+
| features|prediction|
+---------+----------+
|[1.0,0.0]|       1.0|
|[0.0,0.0]|       0.0|
+---------+----------+
...
>>> mlp_path = temp_path + "/mlp"
>>> mlp.save(mlp_path)
>>> mlp2 = MultilayerPerceptronClassifier.load(mlp_path)
>>> mlp2.getBlockSize()
1
>>> model_path = temp_path + "/mlp_model"
>>> model.save(model_path)
>>> model2 = MultilayerPerceptronClassificationModel.load(model_path)
>>> model.layers == model2.layers
True
>>> model.weights == model2.weights
True
>>> mlp2 = mlp2.setInitialWeights(list(range(0, 12)))
>>> model3 = mlp2.fit(df)
>>> model3.weights != model2.weights
True
>>> model3.layers == model.layers
True
```

New in version 1.6.0.

blockSize = *Param(parent='undefined', name='blockSize', doc='Block size for stacking input data in matrices. Data is stacked within partitions. If block size is more than remaining data in a partition then it is adjusted to the size of this data. Recommended size is between 10 and 1000, default is 128.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:

extra – Extra parameters to copy to the new instance

Returns:

Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getBlockSize()

Gets the value of blockSize or its default value.

New in version 1.6.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getInitialWeights()

Gets the value of initialWeights or its default value.

New in version 2.0.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLayers()

Gets the value of layers or its default value.

New in version 1.6.0.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()

Gets the value of probabilityCol or its default value.

getRawPredictionCol()

Gets the value of rawPredictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getSolver()

Gets the value of solver or its default value.

getStepSize()

Gets the value of stepSize or its default value.

New in version 2.0.0.

getThresholds()

Gets the value of thresholds or its default value.

getTol()

Gets the value of tol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

initialWeights = *Param(parent='undefined', name='initialWeights', doc='The initial weights of the model.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

layers = *Param(parent='undefined', name='layers', doc='Sizes of layers from input layer to output layer E.g., Array(780, 100, 10) means 780 inputs, one hidden layer with 100 neurons and output layer of 10 neurons.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setBlockSize(*value*) [source]

Sets the value of **blockSize**.

New in version 1.6.0.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setInitialWeights(*value*) [source]

Sets the value of **initialWeights**.

New in version 2.0.0.

setLabelCol(*value*)

Sets the value of **labelCol**.

New in version 3.0.0.

setLayers(*value*) [source]

Sets the value of **layers**.

New in version 1.6.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

setParams(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *maxIter=100*, *tol=1e-06*, *seed=None*, *layers=None*, *blockSize=128*, *stepSize=0.03*, *solver='l-bfgs'*, *initialWeights=None*, *probabilityCol='probability'*, *rawPredictionCol='rawPrediction'*) [source]

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, tol=1e-6, seed=None, layers=None, blockSize=128, stepSize=0.03, solver="l-bfgs", initialWeights=None, probabilityCol="probability", rawPredictionCol="rawPrediction"): Sets params for MultilayerPerceptronClassifier.

New in version 1.6.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)

Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

setSolver(*value*) [source]

Sets the value of **solver**.

setStepSize(*value*) [source]

Sets the value of **stepSize**.

New in version 2.0.0.

setThresholds(*value*)

Sets the value of **thresholds**.

New in version 3.0.0.

setTol(*value*) [source]

Sets the value of **tol**.

solver = *Param(parent='undefined', name='solver', doc='The solver algorithm for optimization. Supported options: l-bfgs, gd.')*

stepSize = *Param(parent='undefined', name='stepSize', doc='Step size to be used for each iteration of optimization (>= 0).')*

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**MultilayerPerceptronClassificationModel**(*java_model=None*) [source]
Model fitted by MultilayerPerceptronClassifier.

New in version 1.6.0.

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getFeaturesCol()
Gets the value of featuresCol or its default value.

getLabelCol()
Gets the value of labelCol or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getProbabilityCol()
Gets the value of probabilityCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getThresholds()
Gets the value of thresholds or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

property **layers**
array of layer sizes including input and output layers.

New in version 1.6.0.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **numClasses**
Number of classes (values which the label can take).

New in version 2.1.0.

property **numFeatures**

Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)

Predict label for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*)

Sets the value of **probabilityCol**.

New in version 3.0.0.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

New in version 3.0.0.

setThresholds(*value*)

Sets the value of **thresholds**.

New in version 3.0.0.

thresholds = *Param(parent='undefined', name='thresholds', doc="Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class's threshold.")*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

property **weights**

the weights of layers.

New in version 2.0.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**OneVsRest**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *rawPredictionCol='rawPrediction'*, *classifier=None*, *weightCol=None*, *parallelism=1*) [source]

Reduction of Multiclass Classification to Binary Classification. Performs reduction using one against all strategy. For a multiclass classification with k classes, train k models (one per class). Each example is scored against all k models and the model with highest score is picked to label the example.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> data_path = "data/mllib/sample_multiclass_classification_data.txt"
>>> df = spark.read.format("libsvm").load(data_path)
>>> lr = LogisticRegression(regParam=0.01)
>>> ovr = OneVsRest(classifier=lr)
>>> ovr.getRawPredictionCol()
'rawPrediction'
>>> ovr.setPredictionCol("newPrediction")
OneVsRest...
>>> model = ovr.fit(df)
>>> model.models[0].coefficients
DenseVector([0.5..., -1.0..., 3.4..., 4.2...])
>>> model.models[1].coefficients
DenseVector([-2.1..., 3.1..., -2.6..., -2.3...])
>>> model.models[2].coefficients
DenseVector([0.3..., -3.4..., 1.0..., -1.1...])
>>> [x.intercept for x in model.models]
[-2.7..., -2.5..., -1.3...]
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, 0.0, 1.0, 1.0))]).toDF()
>>> model.transform(test0).head().newPrediction
0.0
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(4, [0], [1.0]))]).toDF()
>>> model.transform(test1).head().newPrediction
2.0
>>> test2 = sc.parallelize([Row(features=Vectors.dense(0.5, 0.4, 0.3, 0.2))]).toDF()
>>> model.transform(test2).head().newPrediction
0.0
>>> model_path = temp_path + "/ovr_model"
>>> model.save(model_path)
>>> model2 = OneVsRestModel.load(model_path)
>>> model2.transform(test0).head().newPrediction
0.0
>>> model.transform(test2).columns
['features', 'rawPrediction', 'newPrediction']
```

New in version 2.0.0.

classifier = *Param(parent='undefined', name='classifier', doc='base binary classifier')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*) [source]

Creates a copy of this instance with a randomly generated uid and some extra params. This creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

New in version 2.0.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getClassifier()

Gets the value of classifier or its default value.

New in version 2.0.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParallelism()
Gets the value of parallelism or its default value.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

parallelism = *Param(parent='undefined', name='parallelism', doc='the number of threads to use when running parallel algorithms (>= 1).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setClassifier(*value*) [source]
Sets the value of **classifier**.

New in version 2.0.0.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

setLabelCol(*value*) [source]
Sets the value of **labelCol**.

setParallelism(*value*) [source]
Sets the value of **parallelism**.

setParams(*featuresCol='features', labelCol='label', predictionCol='prediction', rawPredictionCol='rawPrediction', classifier=None, weightCol=None, parallelism=1*) [source]
setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", rawPredictionCol="rawPrediction", classifier=None, weightCol=None, parallelism=1): Sets params for OneVsRest.

New in version 2.0.0.

setPredictionCol(*value*) [source]
Sets the value of **predictionCol**.

setRawPredictionCol(*value*) [source]
Sets the value of **rawPredictionCol**.

setWeightCol(*value*) [source]
Sets the value of **weightCol**.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**OneVsRestModel**(*models*) [source]
Model fitted by OneVsRest. This stores the models resulting from training k binary classifiers: one for each class. Each example is scored against all k models, and the model with the highest score is picked to label the example.

New in version 2.0.0.

classifier = *Param(parent='undefined', name='classifier', doc='base binary classifier')*

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*) [source]
Creates a copy of this instance with a randomly generated uid and some extra params. This creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

New in version 2.0.0.

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getClassifier()
Gets the value of classifier or its default value.

New in version 2.0.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getLabelCol()
Gets the value of labelCol or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getRawPredictionCol()
Gets the value of rawPredictionCol or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setClassifier(*value*) [source]
Sets the value of **classifier**.

New in version 2.0.0.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

setLabelCol(*value*) [source]
Sets the value of **labelCol**.

setPredictionCol(*value*) [source]
Sets the value of **predictionCol**.

setRawPredictionCol(*value*) [source]
Sets the value of **rawPredictionCol**.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

pyspark.ml.clustering module

class pyspark.ml.clustering.**BisectingKMeans**(*featuresCol='features'*, *predictionCol='prediction'*, *maxIter=20*, *seed=None*, *k=4*, *minDivisibleClusterSize=1.0*, *distanceMeasure='euclidean'*) [source]

A bisecting k-means algorithm based on the paper "A comparison of document clustering techniques" by Steinbach, Karypis, and Kumar, with modification to fit Spark. The algorithm starts from a single cluster that contains all points. Iteratively it finds divisible clusters on the bottom level and bisects each of them using k-means, until there are *k* leaf clusters in total or no leaf clusters are divisible. The bisecting steps of clusters on the same level are grouped together to increase parallelism. If bisecting all divisible clusters on the bottom level would result more than *k* leaf clusters, larger clusters get higher priority.

```
>>> from pyspark.ml.linalg import Vectors
>>> data = [(Vectors.dense([0.0, 0.0]),), (Vectors.dense([1.0, 1.0]),),
...         (Vectors.dense([9.0, 8.0]),), (Vectors.dense([8.0, 9.0]),)]
>>> df = spark.createDataFrame(data, ["features"])
>>> bkm = BisectingKMeans(k=2, minDivisibleClusterSize=1.0)
>>> bkm.setMaxIter(10)
BisectingKMeans...
>>> bkm.getMaxIter()
10
>>> bkm.clear(bkm.maxIter)
>>> bkm.setSeed(1)
BisectingKMeans...
>>> bkm.getSeed()
1
>>> bkm.clear(bkm.seed)
>>> model = bkm.fit(df)
>>> model.getMaxIter()
20
>>> model.setPredictionCol("newPrediction")
BisectingKMeansModel...
>>> model.predict(df.head().features)
0
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> model.computeCost(df)
2.0
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
2
>>> summary.clusterSizes
[2, 2]
>>> summary.trainingCost
2.000...
>>> transformed = model.transform(df).select("features", "newPrediction")
>>> rows = transformed.collect()
>>> rows[0].newPrediction == rows[1].newPrediction
True
>>> rows[2].newPrediction == rows[3].newPrediction
True
>>> bkm_path = temp_path + "/bkm"
>>> bkm.save(bkm_path)
>>> bkm2 = BisectingKMeans.load(bkm_path)
>>> bkm2.getK()
2
>>> bkm2.getDistanceMeasure()
'euclidean'
>>> model_path = temp_path + "/bkm_model"
>>> model.save(model_path)
>>> model2 = BisectingKMeansModel.load(model_path)
>>> model2.hasSummary
False
>>> model.clusterCenters()[0] == model2.clusterCenters()[0]
array([ True,  True], dtype=bool)
>>> model.clusterCenters()[1] == model2.clusterCenters()[1]
array([ True,  True], dtype=bool)
```

New in version 2.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance

Returns:
Copy of this instance

distanceMeasure = *Param(parent='undefined', name='distanceMeasure', doc="the distance measure. Supported options: 'euclidean' and 'cosine'.")*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getDistanceMeasure()

Gets the value of distanceMeasure or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getK()

Gets the value of *k* or its default value.

New in version 2.0.0.

getMaxIter()

Gets the value of maxIter or its default value.

getMinDivisibleClusterSize()

Gets the value of *minDivisibleClusterSize* or its default value.

New in version 2.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The desired number of leaf clusters. Must be > 1.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

minDivisibleClusterSize = *Param(parent='undefined', name='minDivisibleClusterSize', doc='The minimum number of points (if >= 1.0) or the minimum proportion of points (if < 1.0) of a divisible cluster.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setDistanceMeasure(*value*) [source]

Sets the value of **distanceMeasure**.

New in version 2.4.0.

setFeaturesCol(*value*) [source]

Sets the value of **featuresCol**.

New in version 2.0.0.

setK(*value*) [source]

Sets the value of **k**.

New in version 2.0.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

New in version 2.0.0.

setMinDivisibleClusterSize(*value*) [source]

Sets the value of **minDivisibleClusterSize**.

New in version 2.0.0.

setParams(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *maxIter=20*, *seed=None*, *k=4*, *minDivisibleClusterSize=1.0*, *distanceMeasure="euclidean"*) [source]

Sets params for BisectingKMeans.

New in version 2.0.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 2.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

New in version 2.0.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.clustering.**BisectingKMeansModel**(*java_model=None*) [source]

Model fitted by BisectingKMeans.

New in version 2.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

clusterCenters() [source]

Get the cluster centers, represented as a list of NumPy arrays.

New in version 2.0.0.

computeCost(*dataset*) [source]

Computes the sum of squared distances between the input points and their corresponding cluster centers.

..note:: Deprecated in 3.0.0. It will be removed in future versions. Use
ClusteringEvaluator instead. You can also get the cost on the training dataset in the summary.

New in version 2.0.0.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

distanceMeasure = *Param(parent='undefined', name='distanceMeasure', doc="the distance measure. Supported options: 'euclidean' and 'cosine'.")*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getDistanceMeasure()

Gets the value of distanceMeasure or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getK()

Gets the value of *k* or its default value.

New in version 2.0.0.

getMaxIter()
Gets the value of maxIter or its default value.

getMinDivisibleClusterSize()
Gets the value of *minDivisibleClusterSize* or its default value.

New in version 2.0.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getSeed()
Gets the value of seed or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

property **hasSummary**
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The desired number of leaf clusters. Must be > 1.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

minDivisibleClusterSize = *Param(parent='undefined', name='minDivisibleClusterSize', doc='The minimum number of points (if >= 1.0) or the minimum proportion of points (if < 1.0) of a divisible cluster.')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*) [source]
Predict label for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*) [source]
Sets the value of **predictionCol**.

New in version 3.0.0.

property **summary**
Gets summary (e.g. cluster assignments, cluster sizes) of the model trained on the training set. An exception is thrown if no summary exists.

New in version 2.1.0.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.clustering.**BisectingKMeansSummary**(*java_obj=None*) [source]
Bisecting KMeans clustering results for a given model.

New in version 2.1.0.

property **cluster**
DataFrame of predicted cluster centers for each training data point.

New in version 2.1.0.

property **clusterSizes**

Size of (number of data points in) each cluster.

New in version 2.1.0.

property **featuresCol**

Name for column of features in *predictions*.

New in version 2.1.0.

property **k**

The number of clusters the model was trained with.

New in version 2.1.0.

property **numIter**

Number of iterations.

New in version 2.4.0.

property **predictionCol**

Name for column of predicted clusters in *predictions*.

New in version 2.1.0.

property **predictions**

DataFrame produced by the model's *transform* method.

New in version 2.1.0.

property **trainingCost**

Sum of squared distances to the nearest centroid for all points in the training dataset. This is equivalent to sklearn's inertia.

New in version 3.0.0.

class pyspark.ml.clustering.**KMeans**(*featuresCol='features'*, *predictionCol='prediction'*, *k=2*, *initMode='k-means||'*, *initSteps=2*, *tol=0.0001*, *maxIter=20*, *seed=None*, *distanceMeasure='euclidean'*, *weightCol=None*) [source]

K-means clustering with a k-means++ like initialization mode (the k-means|| algorithm by Bahmani et al).

```
>>> from pyspark.ml.linalg import Vectors
>>> data = [(Vectors.dense([0.0, 0.0]), 2.0), (Vectors.dense([1.0, 1.0]), 2.0),
...         (Vectors.dense([9.0, 8.0]), 2.0), (Vectors.dense([8.0, 9.0]), 2.0)]
>>> df = spark.createDataFrame(data, ["features", "weighCol"])
>>> kmeans = KMeans(k=2)
>>> kmeans.setSeed(1)
KMeans...
>>> kmeans.setWeightCol("weighCol")
KMeans...
>>> kmeans.setMaxIter(10)
KMeans...
>>> kmeans.getMaxIter()
10
>>> kmeans.clear(kmeans.maxIter)
>>> model = kmeans.fit(df)
>>> model.getDistanceMeasure()
'euclidean'
>>> model.setPredictionCol("newPrediction")
KMeansModel...
>>> model.predict(df.head().features)
0
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> transformed = model.transform(df).select("features", "newPrediction")
>>> rows = transformed.collect()
>>> rows[0].newPrediction == rows[1].newPrediction
True
>>> rows[2].newPrediction == rows[3].newPrediction
True
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
2
>>> summary.clusterSizes
[2, 2]
>>> summary.trainingCost
4.0
>>> kmeans_path = temp_path + "/kmeans"
>>> kmeans.save(kmeans_path)
>>> kmeans2 = KMeans.load(kmeans_path)
>>> kmeans2.getK()
2
>>> model_path = temp_path + "/kmeans_model"
>>> model.save(model_path)
>>> model2 = KMeansModel.load(model_path)
>>> model2.hasSummary
False
>>> model.clusterCenters()[0] == model2.clusterCenters()[0]
array([ True,  True], dtype=bool)
>>> model.clusterCenters()[1] == model2.clusterCenters()[1]
array([ True,  True], dtype=bool)
```

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

distanceMeasure = *Param(parent='undefined', name='distanceMeasure', doc="the distance measure. Supported options: 'euclidean' and 'cosine'.")*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getDistanceMeasure()

Gets the value of distanceMeasure or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getInitMode()

Gets the value of *initMode*

New in version 1.5.0.

getInitSteps()

Gets the value of *initSteps*

New in version 1.5.0.

getK()

Gets the value of *k*

New in version 1.5.0.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getTol()

Gets the value of tol or its default value.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

initMode = *Param(parent='undefined', name='initMode', doc='The initialization algorithm. This can be either "random" to choose random points as initial cluster centers, or "k-means||" to use a parallel variant of k-means++')*

initSteps = *Param(parent='undefined', name='initSteps', doc='The number of steps for k-means|| initialization mode. Must be > 0.')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The number of clusters to create. Must be > 1.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setDistanceMeasure(*value*) [source]

Sets the value of **distanceMeasure**.

New in version 2.4.0.

setFeaturesCol(*value*) [source]

Sets the value of **featuresCol**.

New in version 1.5.0.

setInitMode(*value*) [source]

Sets the value of **initMode**.

New in version 1.5.0.

setInitSteps(*value*) [source]

Sets the value of **initSteps**.

New in version 1.5.0.

setK(*value*) [source]

Sets the value of **k**.

New in version 1.5.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

New in version 1.5.0.

setParams(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *k=2*, *initMode="k-means||"*, *initSteps=2*, *tol=1e-4*, *maxIter=20*, *seed=None*, *distanceMeasure="euclidean"*, *weightCol=None*) [source]

Sets params for KMeans.

New in version 1.5.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 1.5.0.

setSeed(*value*) [source]

Sets the value of **seed**.

New in version 1.5.0.

setTol(*value*) [source]

Sets the value of **tol**.

New in version 1.5.0.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

New in version 3.0.0.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.clustering.**KMeansModel**(*java_model=None*) [source]

Model fitted by KMeans.

New in version 1.5.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

clusterCenters() [source]

Get the cluster centers, represented as a list of NumPy arrays.

New in version 1.5.0.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:

Copy of this instance

distanceMeasure = *Param(parent='undefined', name='distanceMeasure', doc="the distance measure. Supported options: 'euclidean' and 'cosine'.")*

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getDistanceMeasure()
Gets the value of distanceMeasure or its default value.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getInitMode()
Gets the value of *initMode*

New in version 1.5.0.

getInitSteps()
Gets the value of *initSteps*

New in version 1.5.0.

getK()
Gets the value of *k*

New in version 1.5.0.

getMaxIter()
Gets the value of maxIter or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getSeed()
Gets the value of seed or its default value.

getTol()
Gets the value of tol or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

property **hasSummary**
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

initMode = *Param(parent='undefined', name='initMode', doc='The initialization algorithm. This can be either "random" to choose random points as initial cluster centers, or "k-means||" to use a parallel variant of k-means++')*

initSteps = *Param(parent='undefined', name='initSteps', doc='The number of steps for k-means|| initialization mode. Must be > 0.')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The number of clusters to create. Must be > 1.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*) [source]
Predict label for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*) [source]

Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 3.0.0.

property **summary**

Gets summary (e.g. cluster assignments, cluster sizes) of the model trained on the training set. An exception is thrown if no summary exists.

New in version 2.1.0.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an GeneralMLWriter instance for this ML instance.

class pyspark.ml.clustering.**GaussianMixture**(*featuresCol='features'*, *predictionCol='prediction'*, *k=2*, *probabilityCol='probability'*, *tol=0.01*, *maxIter=100*, *seed=None*, *aggregationDepth=2*) [source]

GaussianMixture clustering. This class performs expectation maximization for multivariate Gaussian Mixture Models (GMMs). A GMM represents a composite distribution of independent Gaussian distributions with associated "mixing" weights specifying each's contribution to the composite.

Given a set of sample points, this class will maximize the log-likelihood for a mixture of k Gaussians, iterating until the log-likelihood changes by less than convergenceTol, or until it has reached the max number of iterations. While this process is generally guaranteed to converge, it is not guaranteed to find a global optimum.

Note: For high-dimensional data (with many features), this algorithm may perform poorly. This is due to high-dimensional data (a) making it difficult to cluster at all (based on statistical/theoretical arguments) and (b) numerical issues with Gaussian distributions.

```
>>> from pyspark.ml.linalg import Vectors
```

```
>>> data = [(Vectors.dense([-0.1, -0.05 ]),),
...         (Vectors.dense([-0.01, -0.1]),),
...         (Vectors.dense([0.9, 0.8]),),
...         (Vectors.dense([0.75, 0.935]),),
...         (Vectors.dense([-0.83, -0.68]),),
...         (Vectors.dense([-0.91, -0.76]),)]
>>> df = spark.createDataFrame(data, ["features"])
>>> gm = GaussianMixture(k=3, tol=0.0001, seed=10)
>>> gm.getMaxIter()
100
>>> gm.setMaxIter(10)
GaussianMixture...
>>> gm.getMaxIter()
10
>>> model = gm.fit(df)
>>> model.getAggregationDepth()
2
>>> model.getFeaturesCol()
'features'
>>> model.setPredictionCol("newPrediction")
GaussianMixtureModel...
>>> model.predict(df.head().features)
2
>>> model.predictProbability(df.head().features)
DenseVector([0.0, 0.4736, 0.5264])
>>> model.hasSummary
True
>>> summary = model.summary
>>> summary.k
3
>>> summary.clusterSizes
[2, 2, 2]
>>> summary.logLikelihood
8.14636...
>>> weights = model.weights
>>> len(weights)
3
>>> gaussians = model.gaussians
>>> len(gaussians)
3
>>> model.gaussiansDF.select("mean").head()
Row(mean=DenseVector([0.825, 0.8675]))
>>> model.gaussiansDF.select("cov").head()
Row(cov=DenseMatrix(2, 2, [0.0056, -0.0051, -0.0051, 0.0046], False))
>>> transformed = model.transform(df).select("features", "newPrediction")
>>> rows = transformed.collect()
>>> rows[4].newPrediction == rows[5].newPrediction
True
>>> rows[2].newPrediction == rows[3].newPrediction
True
>>> gmm_path = temp_path + "/gmm"
>>> gm.save(gmm_path)
>>> gm2 = GaussianMixture.load(gmm_path)
>>> gm2.getK()
3
>>> model_path = temp_path + "/gmm_model"
>>> model.save(model_path)
>>> model2 = GaussianMixtureModel.load(model_path)
>>> model2.hasSummary
False
>>> model2.weights == model.weights
True
>>> model2.gaussians == model.gaussians
True
>>> model2.gaussiansDF.select("mean").head()
Row(mean=DenseVector([0.825, 0.8675]))
>>> model2.gaussiansDF.select("cov").head()
Row(cov=DenseMatrix(2, 2, [0.0056, -0.0051, -0.0051, 0.0046], False))
```

New in version 2.0.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of `pyspark.sql.DataFrame`.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getAggregationDepth()
Gets the value of aggregationDepth or its default value.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getK()
Gets the value of *k*

New in version 2.0.0.

getMaxIter()
Gets the value of maxIter or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getProbabilityCol()
Gets the value of probabilityCol or its default value.

getSeed()
Gets the value of seed or its default value.

getTol()
Gets the value of tol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='Number of independent Gaussians in the mixture model. Must be > 1.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**
Returns all params ordered by name. The default implementation uses `dir()` to get all attributes of type `Param`.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setAggregationDepth(*value*) [source]
Sets the value of `aggregationDepth`.

New in version 3.0.0.

setFeaturesCol(*value*) [source]
Sets the value of `featuresCol`.

New in version 2.0.0.

setK(*value*) [source]
Sets the value of `k`.

New in version 2.0.0.

setMaxIter(*value*) [source]
Sets the value of `maxIter`.

New in version 2.0.0.

setParams(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *k=2*, *probabilityCol="probability"*, *tol=0.01*,

maxIter=100, seed=None, aggregationDepth=2) [source]

Sets params for GaussianMixture.

New in version 2.0.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 2.0.0.

setProbabilityCol(*value*) [source]

Sets the value of **probabilityCol**.

New in version 2.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

New in version 2.0.0.

setTol(*value*) [source]

Sets the value of **tol**.

New in version 2.0.0.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.clustering.**GaussianMixtureModel**(*java_model=None*) [source]

Model fitted by GaussianMixture.

New in version 2.0.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

property **gaussians**

Array of **MultivariateGaussian** where gaussians[i] represents the Multivariate Gaussian (Normal) Distribution for Gaussian i

New in version 3.0.0.

property **gaussiansDF**

Retrieve Gaussian distributions as a DataFrame. Each row represents a Gaussian Distribution. The DataFrame has two columns: mean (Vector) and cov (Matrix).

New in version 2.0.0.

getAggregationDepth()

Gets the value of aggregationDepth or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getK()

Gets the value of *k*

New in version 2.0.0.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getProbabilityCol()

Gets the value of probabilityCol or its default value.

getSeed()
Gets the value of seed or its default value.

getTol()
Gets the value of tol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

property **hasSummary**
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='Number of independent Gaussians in the mixture model. Must be > 1.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*) [source]
Predict label for the given features.

New in version 3.0.0.

predictProbability(*value*) [source]
Predict probability for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*) [source]
Sets the value of **predictionCol**.

New in version 3.0.0.

setProbabilityCol(*value*) [source]
Sets the value of **probabilityCol**.

New in version 3.0.0.

property **summary**
Gets summary (e.g. cluster assignments, cluster sizes) of the model trained on the training set. An exception is thrown if no summary exists.

New in version 2.1.0.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

property **weights**
Weight for each Gaussian distribution in the mixture. This is a multinomial probability distribution over the k Gaussians, where weights[i] is the weight for Gaussian i, and weights sum to 1.

New in version 2.0.0.

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.clustering.**GaussianMixtureSummary**(*java_obj=None*) [source]

Gaussian mixture clustering results for a given model.

New in version 2.1.0.

property **cluster**

DataFrame of predicted cluster centers for each training data point.

New in version 2.1.0.

property **clusterSizes**

Size of (number of data points in) each cluster.

New in version 2.1.0.

property **featuresCol**

Name for column of features in *predictions*.

New in version 2.1.0.

property **k**

The number of clusters the model was trained with.

New in version 2.1.0.

property **logLikelihood**

Total log-likelihood for this model on the given data.

New in version 2.2.0.

property **numIter**

Number of iterations.

New in version 2.4.0.

property **predictionCol**

Name for column of predicted clusters in *predictions*.

New in version 2.1.0.

property **predictions**

DataFrame produced by the model's *transform* method.

New in version 2.1.0.

property **probability**

DataFrame of probabilities of each cluster for each training data point.

New in version 2.1.0.

property **probabilityCol**

Name for column of predicted probability of each cluster in *predictions*.

New in version 2.1.0.

class pyspark.ml.clustering.**LDA**(*featuresCol='features', maxIter=20, seed=None, checkpointInterval=10, k=10, optimizer='online', learningOffset=1024.0, learningDecay=0.51, subsamplingRate=0.05, optimizeDocConcentration=True, docConcentration=None, topicConcentration=None, topicDistributionCol='topicDistribution', keepLastCheckpoint=True*) [source]

Latent Dirichlet Allocation (LDA), a topic model designed for text documents.

Terminology:

- "term" = "word": an element of the vocabulary
- "token": instance of a term appearing in a document
- "topic": multinomial distribution over terms representing some concept
- "document": one piece of text, corresponding to one row in the input data

Original LDA paper (journal version):

Blei, Ng, and Jordan. "Latent Dirichlet Allocation." JMLR, 2003.

Input data (featuresCol): LDA is given a collection of documents as input data, via the featuresCol parameter. Each document is specified as a **Vector** of length vocabSize, where each entry is the count for the corresponding term (word) in the document. Feature transformers such as **pyspark.ml.feature.Tokenizer** and **pyspark.ml.feature.CountVectorizer** can be useful for converting text to word count vectors.

```
>>> from pyspark.ml.linalg import Vectors, SparseVector
>>> from pyspark.ml.clustering import LDA
>>> df = spark.createDataFrame([[1, Vectors.dense([0.0, 1.0])],
...      [2, SparseVector(2, {0: 1.0})],], ["id", "features"])
>>> lda = LDA(k=2, seed=1, optimizer="em")
>>> lda.setMaxIter(10)
LDA...
>>> lda.getMaxIter()
10
>>> lda.clear(lda.maxIter)
>>> model = lda.fit(df)
>>> model.setSeed(1)
DistributedLDAModel...
>>> model.getTopicDistributionCol()
'topicDistribution'
>>> model.isDistributed()
True
>>> localModel = model.toLocal()
>>> localModel.isDistributed()
False
>>> model.vocabSize()
2
>>> model.describeTopics().show()
+-----+-----------+--------------------+
|topic|termIndices|         termWeights|
+-----+-----------+--------------------+
|    0|     [1, 0]|[0.50401530077160...|
|    1|     [0, 1]|[0.50401530077160...|
+-----+-----------+--------------------+
...
>>> model.topicsMatrix()
DenseMatrix(2, 2, [0.496, 0.504, 0.504, 0.496], 0)
>>> lda_path = temp_path + "/lda"
>>> lda.save(lda_path)
>>> sameLDA = LDA.load(lda_path)
>>> distributed_model_path = temp_path + "/lda_distributed_model"
>>> model.save(distributed_model_path)
>>> sameModel = DistributedLDAModel.load(distributed_model_path)
>>> local_model_path = temp_path + "/lda_local_model"
>>> localModel.save(local_model_path)
>>> sameLocalModel = LocalLDAModel.load(local_model_path)
```

New in version 2.0.0.

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

docConcentration = *Param(parent='undefined', name='docConcentration', doc='Concentration parameter (commonly named "alpha") for the prior placed on documents\' distributions over topics ("theta").')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getDocConcentration()

Gets the value of **docConcentration** or its default value.

New in version 2.0.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getK()
Gets the value of **k** or its default value.

New in version 2.0.0.

getKeepLastCheckpoint()
Gets the value of **keepLastCheckpoint** or its default value.

New in version 2.0.0.

getLearningDecay()
Gets the value of **learningDecay** or its default value.

New in version 2.0.0.

getLearningOffset()
Gets the value of **learningOffset** or its default value.

New in version 2.0.0.

getMaxIter()
Gets the value of maxIter or its default value.

getOptimizeDocConcentration()
Gets the value of **optimizeDocConcentration** or its default value.

New in version 2.0.0.

getOptimizer()
Gets the value of **optimizer** or its default value.

New in version 2.0.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getSeed()
Gets the value of seed or its default value.

getSubsamplingRate()
Gets the value of **subsamplingRate** or its default value.

New in version 2.0.0.

getTopicConcentration()
Gets the value of **topicConcentration** or its default value.

New in version 2.0.0.

getTopicDistributionCol()
Gets the value of **topicDistributionCol** or its default value.

New in version 2.0.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The number of topics (clusters) to infer. Must be > 1.')*

keepLastCheckpoint = *Param(parent='undefined', name='keepLastCheckpoint', doc='(For EM optimizer) If using checkpointing, this indicates whether to keep the last checkpoint. If false, then the checkpoint will be deleted. Deleting the checkpoint can cause failures if a data partition is lost, so set this bit with care.')*

learningDecay = *Param(parent='undefined', name='learningDecay', doc='Learning rate, set as anexponential decay rate. This should be between (0.5, 1.0] to guarantee asymptotic convergence.')*

learningOffset = *Param(parent='undefined', name='learningOffset', doc='A (positive) learning parameter that downweights early iterations. Larger values make early iterations count less')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

optimizeDocConcentration = *Param(parent='undefined', name='optimizeDocConcentration', doc='Indicates whether the docConcentration (Dirichlet parameter for document-topic distribution) will be optimized during training.')*

optimizer = *Param(parent='undefined', name='optimizer', doc='Optimizer or inference algorithm used to estimate the LDA model. Supported: online, em')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setCheckpointInterval(*value*) [source]

Sets the value of **checkpointInterval**.

New in version 2.0.0.

setDocConcentration(*value*) [source]

Sets the value of **docConcentration**.

```
>>> algo = LDA().setDocConcentration([0.1, 0.2])
>>> algo.getDocConcentration()
[0.1..., 0.2...]
```

New in version 2.0.0.

setFeaturesCol(*value*) [source]

Sets the value of **featuresCol**.

New in version 2.0.0.

setK(*value*) [source]

Sets the value of **k**.

```
>>> algo = LDA().setK(10)
>>> algo.getK()
10
```

New in version 2.0.0.

setKeepLastCheckpoint(*value*) [source]

Sets the value of **keepLastCheckpoint**.

```
>>> algo = LDA().setKeepLastCheckpoint(False)
>>> algo.getKeepLastCheckpoint()
False
```

New in version 2.0.0.

setLearningDecay(*value*) [source]

Sets the value of **learningDecay**.

```
>>> algo = LDA().setLearningDecay(0.1)
>>> algo.getLearningDecay()
0.1...
```

New in version 2.0.0.

setLearningOffset(*value*) [source]

Sets the value of **learningOffset**.

```
>>> algo = LDA().setLearningOffset(100)
>>> algo.getLearningOffset()
100.0
```

New in version 2.0.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

New in version 2.0.0.

setOptimizeDocConcentration(*value*) [source]

Sets the value of **optimizeDocConcentration**.

```
>>> algo = LDA().setOptimizeDocConcentration(True)
>>> algo.getOptimizeDocConcentration()
True
```

New in version 2.0.0.

setOptimizer(*value*) [source]

Sets the value of **optimizer**. Currently only support 'em' and 'online'.

```
>>> algo = LDA().setOptimizer("em")
>>> algo.getOptimizer()
'em'
```

New in version 2.0.0.

setParams(*self*, *featuresCol="features"*, *maxIter=20*, *seed=None*, *checkpointInterval=10*, *k=10*, *optimizer="online"*, *learningOffset=1024.0*, *learningDecay=0.51*, *subsamplingRate=0.05*, *optimizeDocConcentration=True*, *docConcentration=None*, *topicConcentration=None*, *topicDistributionCol="topicDistribution"*, *keepLastCheckpoint=True*)

Sets params for LDA. [source]

New in version 2.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

New in version 2.0.0.

setSubsamplingRate(*value*) [source]

Sets the value of **subsamplingRate**.

```
>>> algo = LDA().setSubsamplingRate(0.1)
>>> algo.getSubsamplingRate()
0.1...
```

New in version 2.0.0.

setTopicConcentration(*value*) [source]

Sets the value of **topicConcentration**.

```
>>> algo = LDA().setTopicConcentration(0.5)
>>> algo.getTopicConcentration()
0.5...
```

New in version 2.0.0.

setTopicDistributionCol(*value*) [source]

Sets the value of **topicDistributionCol**.

```
>>> algo = LDA().setTopicDistributionCol("topicDistributionCol")
>>> algo.getTopicDistributionCol()
'topicDistributionCol'
```

New in version 2.0.0.

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the corpus to be sampled and used in each iteration of mini-batch gradient descent, in range (0, 1].')*

topicConcentration = *Param(parent='undefined', name='topicConcentration', doc='Concentration parameter (commonly named "beta" or "eta") for the prior placed on topic\' distributions over terms.')*

topicDistributionCol = *Param(parent='undefined', name='topicDistributionCol', doc='Output column with estimates of the topic mixture distribution for each document (often called "theta" in the literature). Returns a vector of zeros for an empty document.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.clustering.**LDAModel**(*java_model=None*) [source]

Latent Dirichlet Allocation (LDA) model. This abstraction permits for different underlying representations, including local and distributed data structures.

New in version 2.0.0.

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

describeTopics(*maxTermsPerTopic=10*) [source]

Return the topics described by their top-weighted terms.

New in version 2.0.0.

docConcentration = *Param(parent='undefined', name='docConcentration', doc='Concentration parameter (commonly named "alpha") for the prior placed on documents\' distributions over topics ("theta").')*

estimatedDocConcentration() [source]

Value for **LDA.docConcentration** estimated from data. If Online LDA was used and **LDA.optimizeDocConcentration** was set to false, then this returns the fixed (given) value for the **LDA.docConcentration** parameter.

New in version 2.0.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getDocConcentration()

Gets the value of **docConcentration** or its default value.

New in version 2.0.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getK()
Gets the value of **k** or its default value.

New in version 2.0.0.

getKeepLastCheckpoint()
Gets the value of **keepLastCheckpoint** or its default value.

New in version 2.0.0.

getLearningDecay()
Gets the value of **learningDecay** or its default value.

New in version 2.0.0.

getLearningOffset()
Gets the value of **learningOffset** or its default value.

New in version 2.0.0.

getMaxIter()
Gets the value of maxIter or its default value.

getOptimizeDocConcentration()
Gets the value of **optimizeDocConcentration** or its default value.

New in version 2.0.0.

getOptimizer()
Gets the value of **optimizer** or its default value.

New in version 2.0.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getSeed()
Gets the value of seed or its default value.

getSubsamplingRate()
Gets the value of **subsamplingRate** or its default value.

New in version 2.0.0.

getTopicConcentration()
Gets the value of **topicConcentration** or its default value.

New in version 2.0.0.

getTopicDistributionCol()
Gets the value of **topicDistributionCol** or its default value.

New in version 2.0.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isDistributed() [source]
Indicates whether this instance is of type DistributedLDAModel

New in version 2.0.0.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The number of topics (clusters) to infer. Must be > 1.')*

keepLastCheckpoint = *Param(parent='undefined', name='keepLastCheckpoint', doc='(For EM optimizer) If using checkpointing, this indicates whether to keep the last checkpoint. If false, then the checkpoint will be deleted. Deleting the checkpoint can cause failures if a data partition is lost, so set this bit with care.')*

learningDecay = *Param(parent='undefined', name='learningDecay', doc='Learning rate, set as anexponential decay rate. This should be between (0.5, 1.0] to guarantee asymptotic convergence.')*

learningOffset = *Param(parent='undefined', name='learningOffset', doc='A (positive) learning parameter that downweights early iterations. Larger values make early iterations count less')*

logLikelihood(*dataset*) [source]
Calculates a lower bound on the log likelihood of the entire corpus. See Equation (16) in the Online LDA paper (Hoffman et al., 2010).

WARNING: If this model is an instance of **DistributedLDAModel** (produced when **optimizer** is set to “em”), this involves collecting a large **topicsMatrix()** to the driver. This implementation may be changed in the future.

New in version 2.0.0.

logPerplexity(*dataset*) [source]
Calculate an upper bound on perplexity. (Lower is better.) See Equation (16) in the Online LDA paper (Hoffman et al., 2010).

WARNING: If this model is an instance of **DistributedLDAModel** (produced when **optimizer** is set to “em”), this involves collecting a large **topicsMatrix()** to the driver. This implementation may be changed in the future.

New in version 2.0.0.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

optimizeDocConcentration = *Param(parent='undefined', name='optimizeDocConcentration', doc='Indicates whether the docConcentration (Dirichlet parameter for document-topic distribution) will be optimized during training.')*

optimizer = *Param(parent='undefined', name='optimizer', doc='Optimizer or inference algorithm used to estimate the LDA model. Supported: online, em')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*) [source]

Sets the value of **featuresCol**.

New in version 3.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

New in version 3.0.0.

setTopicDistributionCol(*value*) [source]

Sets the value of **topicDistributionCol**.

```
>>> algo = LDA().setTopicDistributionCol("topicDistributionCol")
>>> algo.getTopicDistributionCol()
'topicDistributionCol'
```

New in version 3.0.0.

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the corpus to be sampled and used in each iteration of mini-batch gradient descent, in range (0, 1].')*

topicConcentration = *Param(parent='undefined', name='topicConcentration', doc='Concentration parameter (commonly named "beta" or "eta") for the prior placed on topic\' distributions over terms.')*

topicDistributionCol = *Param(parent='undefined', name='topicDistributionCol', doc='Output column with estimates of the topic mixture distribution for each document (often called "theta" in the literature). Returns a vector of zeros for an empty document.')*

topicsMatrix() [source]

Inferred topics, where each topic is represented by a distribution over terms. This is a matrix of size vocabSize x k, where each column is a topic. No guarantees are given about the ordering of the topics.

WARNING: If this model is actually a **DistributedLDAModel** instance produced by the Expectation-Maximization ("em") *optimizer*, then this method could involve collecting a large amount of data to the driver (on the order of vocabSize x k).

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

vocabSize() [source]

Vocabulary size (number of terms or words in the vocabulary)

New in version 2.0.0.

class pyspark.ml.clustering.**LocalLDAModel**(*java_model=None*) [source]

Local (non-distributed) model fitted by **LDA**. This model stores the inferred topics only; it does not store info about the training dataset.

New in version 2.0.0.

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

describeTopics(*maxTermsPerTopic=10*)

Return the topics described by their top-weighted terms.

New in version 2.0.0.

docConcentration = *Param(parent='undefined', name='docConcentration', doc='Concentration parameter (commonly named "alpha") for the prior placed on documents\' distributions over topics ("theta").')*

estimatedDocConcentration()

Value for **LDA.docConcentration** estimated from data. If Online LDA was used and **LDA.optimizeDocConcentration** was set to false, then this returns the fixed (given) value for the **LDA.docConcentration** parameter.

New in version 2.0.0.

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCheckpointInterval()
Gets the value of checkpointInterval or its default value.

getDocConcentration()
Gets the value of **docConcentration** or its default value.

New in version 2.0.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getK()
Gets the value of **k** or its default value.

New in version 2.0.0.

getKeepLastCheckpoint()
Gets the value of **keepLastCheckpoint** or its default value.

New in version 2.0.0.

getLearningDecay()
Gets the value of **learningDecay** or its default value.

New in version 2.0.0.

getLearningOffset()
Gets the value of **learningOffset** or its default value.

New in version 2.0.0.

getMaxIter()
Gets the value of maxIter or its default value.

getOptimizeDocConcentration()
Gets the value of **optimizeDocConcentration** or its default value.

New in version 2.0.0.

getOptimizer()
Gets the value of **optimizer** or its default value.

New in version 2.0.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getSeed()
Gets the value of seed or its default value.

getSubsamplingRate()
Gets the value of **subsamplingRate** or its default value.

New in version 2.0.0.

getTopicConcentration()
Gets the value of **topicConcentration** or its default value.

New in version 2.0.0.

getTopicDistributionCol()
Gets the value of **topicDistributionCol** or its default value.

New in version 2.0.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isDistributed()
Indicates whether this instance is of type DistributedLDAModel

New in version 2.0.0.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The number of topics (clusters) to infer. Must be > 1.')*

keepLastCheckpoint = *Param(parent='undefined', name='keepLastCheckpoint', doc='(For EM optimizer) If using checkpointing, this indicates whether to keep the last checkpoint. If false, then the checkpoint will be deleted. Deleting the checkpoint can cause failures if a data partition is lost, so set this bit with care.')*

learningDecay = *Param(parent='undefined', name='learningDecay', doc='Learning rate, set as anexponential decay rate. This should be between (0.5, 1.0] to guarantee asymptotic convergence.')*

learningOffset = *Param(parent='undefined', name='learningOffset', doc='A (positive) learning parameter that downweights early iterations. Larger values make early iterations count less')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

logLikelihood(*dataset*)

Calculates a lower bound on the log likelihood of the entire corpus. See Equation (16) in the Online LDA paper (Hoffman et al., 2010).

WARNING: If this model is an instance of **DistributedLDAModel** (produced when **optimizer** is set to “em”), this involves collecting a large **topicsMatrix()** to the driver. This implementation may be changed in the future.

New in version 2.0.0.

logPerplexity(*dataset*)

Calculate an upper bound on perplexity. (Lower is better.) See Equation (16) in the Online LDA paper (Hoffman et al., 2010).

WARNING: If this model is an instance of **DistributedLDAModel** (produced when **optimizer** is set to “em”), this involves collecting a large **topicsMatrix()** to the driver. This implementation may be changed in the future.

New in version 2.0.0.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

optimizeDocConcentration = *Param(parent='undefined', name='optimizeDocConcentration', doc='Indicates whether the docConcentration (Dirichlet parameter for document-topic distribution) will be optimized during training.')*

optimizer = *Param(parent='undefined', name='optimizer', doc='Optimizer or inference algorithm used to estimate the LDA model. Supported: online, em')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of ‘write().save(path)’.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setSeed(*value*)

Sets the value of **seed**.

New in version 3.0.0.

setTopicDistributionCol(*value*)

Sets the value of **topicDistributionCol**.

```
>>> algo = LDA().setTopicDistributionCol("topicDistributionCol")
>>> algo.getTopicDistributionCol()
'topicDistributionCol'
```

New in version 3.0.0.

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the corpus to be sampled and used in each iteration of mini-batch gradient descent, in range (0, 1].')*

topicConcentration = *Param(parent='undefined', name='topicConcentration', doc='Concentration parameter (commonly named "beta" or "eta") for the prior placed on topic\' distributions over terms.')*

topicDistributionCol = *Param(parent='undefined', name='topicDistributionCol', doc='Output column with estimates of the topic mixture distribution for each document (often called "theta" in the literature). Returns a vector of zeros for an empty document.')*

topicsMatrix()

Inferred topics, where each topic is represented by a distribution over terms. This is a matrix of size vocabSize x k, where each column is a topic. No guarantees are given about the ordering of the topics.

WARNING: If this model is actually a **DistributedLDAModel** instance produced by the Expectation-Maximization (“em”) *optimizer*, then this method could involve collecting a large amount of data to the driver (on the order of vocabSize x k).

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

vocabSize()

Vocabulary size (number of terms or words in the vocabulary)

New in version 2.0.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.clustering.**DistributedLDAModel**(*java_model=None*) [source]

Distributed model fitted by **LDA**. This type of model is currently only produced by Expectation-Maximization (EM).

This model stores the inferred topics, the full training dataset, and the topic distribution for each training document.

New in version 2.0.0.

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

describeTopics(*maxTermsPerTopic=10*)

Return the topics described by their top-weighted terms.

New in version 2.0.0.

docConcentration = *Param(parent='undefined', name='docConcentration', doc='Concentration parameter (commonly named "alpha") for the prior placed on documents\' distributions over topics ("theta").')*

estimatedDocConcentration()

Value for **LDA.docConcentration** estimated from data. If Online LDA was used and **LDA.optimizeDocConcentration** was set to false, then this returns the fixed (given) value for the **LDA.docConcentration** parameter.

New in version 2.0.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCheckpointFiles() [source]

If using checkpointing and **LDA.keepLastCheckpoint** is set to true, then there may be saved checkpoint files. This method is provided so that users can manage those files.

Note: Removing the checkpoints can cause failures if a partition is lost and is needed by certain **DistributedLDAModel** methods. Reference counting will clean up the checkpoints when this model and derivative data go out of scope.

:return List of checkpoint files from training

New in version 2.0.0.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getDocConcentration()

Gets the value of **docConcentration** or its default value.

New in version 2.0.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getK()

Gets the value of **k** or its default value.

New in version 2.0.0.

getKeepLastCheckpoint()

Gets the value of **keepLastCheckpoint** or its default value.

New in version 2.0.0.

getLearningDecay()

Gets the value of **learningDecay** or its default value.

New in version 2.0.0.

getLearningOffset()

Gets the value of **learningOffset** or its default value.

New in version 2.0.0.

getMaxIter()
Gets the value of maxIter or its default value.

getOptimizeDocConcentration()
Gets the value of **optimizeDocConcentration** or its default value.

New in version 2.0.0.

getOptimizer()
Gets the value of **optimizer** or its default value.

New in version 2.0.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getSeed()
Gets the value of seed or its default value.

getSubsamplingRate()
Gets the value of **subsamplingRate** or its default value.

New in version 2.0.0.

getTopicConcentration()
Gets the value of **topicConcentration** or its default value.

New in version 2.0.0.

getTopicDistributionCol()
Gets the value of **topicDistributionCol** or its default value.

New in version 2.0.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isDistributed()
Indicates whether this instance is of type DistributedLDAModel

New in version 2.0.0.

isSet(*param*)
Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The number of topics (clusters) to infer. Must be > 1.')*

keepLastCheckpoint = *Param(parent='undefined', name='keepLastCheckpoint', doc='(For EM optimizer) If using checkpointing, this indicates whether to keep the last checkpoint. If false, then the checkpoint will be deleted. Deleting the checkpoint can cause failures if a data partition is lost, so set this bit with care.')*

learningDecay = *Param(parent='undefined', name='learningDecay', doc='Learning rate, set as anexponential decay rate. This should be between (0.5, 1.0] to guarantee asymptotic convergence.')*

learningOffset = *Param(parent='undefined', name='learningOffset', doc='A (positive) learning parameter that downweights early iterations. Larger values make early iterations count less')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

logLikelihood(*dataset*)
Calculates a lower bound on the log likelihood of the entire corpus. See Equation (16) in the Online LDA paper (Hoffman et al., 2010).

WARNING: If this model is an instance of **DistributedLDAModel** (produced when **optimizer** is set to “em”), this involves collecting a large **topicsMatrix()** to the driver. This implementation may be changed in the future.

New in version 2.0.0.

logPerplexity(*dataset*)
Calculate an upper bound on perplexity. (Lower is better.) See Equation (16) in the Online LDA paper (Hoffman et al., 2010).

WARNING: If this model is an instance of **DistributedLDAModel** (produced when **optimizer** is set to “em”), this involves collecting a large **topicsMatrix()** to the driver. This implementation may be changed in the future.

New in version 2.0.0.

logPrior() [source]
Log probability of the current parameter estimate: log P(topics, topic distributions for docs | alpha, eta)

New in version 2.0.0.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

optimizeDocConcentration = *Param(parent='undefined', name='optimizeDocConcentration', doc='Indicates whether the docConcentration (Dirichlet parameter for document-topic distribution) will be optimized during training.')*

optimizer = *Param(parent='undefined', name='optimizer', doc='Optimizer or inference algorithm used to estimate the LDA model. Supported: online, em')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setSeed(*value*)

Sets the value of **seed**.

New in version 3.0.0.

setTopicDistributionCol(*value*)

Sets the value of **topicDistributionCol**.

```
>>> algo = LDA().setTopicDistributionCol("topicDistributionCol")
>>> algo.getTopicDistributionCol()
'topicDistributionCol'
```

New in version 3.0.0.

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the corpus to be sampled and used in each iteration of mini-batch gradient descent, in range (0, 1].')*

toLocal() [source]

Convert this distributed model to a local representation. This discards info about the training dataset.

WARNING: This involves collecting a large **topicsMatrix()** to the driver.

New in version 2.0.0.

topicConcentration = *Param(parent='undefined', name='topicConcentration', doc='Concentration parameter (commonly named "beta" or "eta") for the prior placed on topic\' distributions over terms.')*

topicDistributionCol = *Param(parent='undefined', name='topicDistributionCol', doc='Output column with estimates of the topic mixture distribution for each document (often called "theta" in the literature). Returns a vector of zeros for an empty document.')*

topicsMatrix()

Inferred topics, where each topic is represented by a distribution over terms. This is a matrix of size vocabSize x k, where each column is a topic. No guarantees are given about the ordering of the topics.

WARNING: If this model is actually a **DistributedLDAModel** instance produced by the Expectation-Maximization ("em") *optimizer*, then this method could involve collecting a large amount of data to the driver (on the order of vocabSize x k).

New in version 2.0.0.

trainingLogLikelihood() [source]

Log likelihood of the observed tokens in the training set, given the current parameter estimates: log P(docs | topics, topic distributions for docs, Dirichlet hyperparameters)

Notes:

- This excludes the prior; for that, use **logPrior()**.
- Even with **logPrior()**, this is NOT the same as the data log likelihood given the hyperparameters.
- This is computed from the topic distributions computed during training. If you call **logLikelihood()** on the same training dataset, the topic distributions will be computed again, possibly giving different results.

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

vocabSize()

Vocabulary size (number of terms or words in the vocabulary)

New in version 2.0.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.clustering.**PowerIterationClustering**(*k=2*, *maxIter=20*, *initMode='random'*, *srcCol='src'*, *dstCol='dst'*, *weightCol=None*) [source]

Power Iteration Clustering (PIC), a scalable graph clustering algorithm developed by Lin and Cohen. From the abstract: PIC finds a very low-dimensional embedding of a dataset using truncated power iteration on a normalized pair-wise similarity matrix of the data.

This class is not yet an Estimator/Transformer, use **assignClusters()** method to run the PowerIterationClustering algorithm.

See also: Wikipedia on Spectral clustering

```
>>> data = [(1, 0, 0.5),
...         (2, 0, 0.5), (2, 1, 0.7),
...         (3, 0, 0.5), (3, 1, 0.7), (3, 2, 0.9),
...         (4, 0, 0.5), (4, 1, 0.7), (4, 2, 0.9), (4, 3, 1.1),
...         (5, 0, 0.5), (5, 1, 0.7), (5, 2, 0.9), (5, 3, 1.1), (5, 4, 1.3)]
>>> df = spark.createDataFrame(data).toDF("src", "dst", "weight").repartition(1)
>>> pic = PowerIterationClustering(k=2, weightCol="weight")
>>> pic.setMaxIter(40)
PowerIterationClustering...
>>> assignments = pic.assignClusters(df)
>>> assignments.sort(assignments.id).show(truncate=False)
+---+-------+
|id |cluster|
+---+-------+
|0  |0      |
|1  |0      |
|2  |0      |
|3  |0      |
|4  |0      |
|5  |1      |
+---+-------+
...
>>> pic_path = temp_path + "/pic"
>>> pic.save(pic_path)
>>> pic2 = PowerIterationClustering.load(pic_path)
>>> pic2.getK()
2
>>> pic2.getMaxIter()
40
```

New in version 2.4.0.

assignClusters(*dataset*) [source]

Run the PIC algorithm and returns a cluster assignment for each input vertex.

Parameters:
dataset – A dataset with columns src, dst, weight representing the affinity matrix, which is the matrix A in the PIC paper. Suppose the src column value is i, the dst column value is j, the weight column value is similarity s,,ij,, which must be nonnegative. This is a symmetric matrix and hence s,,ij,, = s,,ji,,. For any (i, j) with nonzero similarity, there should be either (i, j, s,,ij,,) or (j, i, s,,ji,,) in the input. Rows with i = j are ignored, because we assume s,,ij,, = 0.0.
Returns:
A dataset that contains columns of vertex id and the corresponding cluster for the id. The schema of it will be: - id: Long - cluster: Int

New in version 2.4.0.

New in version 2.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

dstCol = *Param(parent='undefined', name='dstCol', doc='Name of the input column for destination vertex IDs.')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getDstCol()

Gets the value of **dstCol** or its default value.

New in version 2.4.0.

getInitMode()

Gets the value of **initMode** or its default value.

New in version 2.4.0.

getK()

Gets the value of **k** or its default value.

New in version 2.4.0.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getSrcCol()

Gets the value of **srcCol** or its default value.

New in version 2.4.0.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

initMode = *Param(parent='undefined', name='initMode', doc="The initialization algorithm. This can be either 'random' to use a random vector as vertex properties, or 'degree' to use a normalized sum of similarities with other vertices. Supported options: 'random' and 'degree'.")*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The number of clusters to create. Must be > 1.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setDstCol(*value*) [source]

Sets the value of **dstCol**.

New in version 2.4.0.

setInitMode(*value*) [source]

Sets the value of **initMode**.

New in version 2.4.0.

setK(*value*) [source]

Sets the value of **k**.

New in version 2.4.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

New in version 2.4.0.

setParams(*self*, *k=2*, *maxIter=20*, *initMode="random"*, *srcCol="src"*, *dstCol="dst"*, *weightCol=None*) [source]

Sets params for PowerIterationClustering.

New in version 2.4.0.

setSrcCol(*value*) [source]

Sets the value of **srcCol**.

New in version 2.4.0.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

New in version 2.4.0.

srcCol = *Param(parent='undefined', name='srcCol', doc='Name of the input column for source vertex IDs.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

pyspark.ml.linalg module

MLlib utilities for linear algebra. For dense vectors, MLlib uses the NumPy *array* type, so you can simply pass NumPy arrays around. For sparse vectors, users can construct a **SparseVector** object from MLlib or pass SciPy *scipy.sparse* column vectors if SciPy is available in their environment.

class pyspark.ml.linalg.**Vector** [source]

toArray() [source]

Convert the vector into an numpy.ndarray

Returns:
numpy.ndarray

class pyspark.ml.linalg.**DenseVector**(*ar*) [source]

A dense vector represented by a value array. We use numpy array for storage and arithmetics will be delegated to the underlying numpy array.

```
>>> v = Vectors.dense([1.0, 2.0])
>>> u = Vectors.dense([3.0, 4.0])
>>> v + u
DenseVector([4.0, 6.0])
>>> 2 - v
DenseVector([1.0, 0.0])
>>> v / 2
DenseVector([0.5, 1.0])
>>> v * u
DenseVector([3.0, 8.0])
>>> u / v
DenseVector([3.0, 2.0])
>>> u % 2
DenseVector([1.0, 0.0])
>>> -v
DenseVector([-1.0, -2.0])
```

dot(*other*) [source]

Compute the dot product of two Vectors. We support (Numpy array, list, SparseVector, or SciPy sparse) and a target NumPy array that is either 1- or 2-dimensional. Equivalent to calling numpy.dot of the two vectors.

```
>>> dense = DenseVector(array.array('d', [1., 2.]))
>>> dense.dot(dense)
5.0
>>> dense.dot(SparseVector(2, [0, 1], [2., 1.]))
4.0
>>> dense.dot(range(1, 3))
5.0
>>> dense.dot(np.array(range(1, 3)))
5.0
>>> dense.dot([1.,])
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
>>> dense.dot(np.reshape([1., 2., 3., 4.], (2, 2), order='F'))
array([  5.,  11.])
>>> dense.dot(np.reshape([1., 2., 3.], (3, 1), order='F'))
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
```

norm(*p*) [source]

Calculates the norm of a DenseVector.

```
>>> a = DenseVector([0, -1, 2, -3])
>>> a.norm(2)
3.7...
>>> a.norm(1)
6.0
```

numNonzeros() [source]

Number of nonzero elements. This scans all active values and count non zeros

squared_distance(*other*) [source]

Squared distance of two Vectors.

```
>>> dense1 = DenseVector(array.array('d', [1., 2.]))
>>> dense1.squared_distance(dense1)
0.0
>>> dense2 = np.array([2., 1.])
>>> dense1.squared_distance(dense2)
2.0
>>> dense3 = [2., 1.]
>>> dense1.squared_distance(dense3)
2.0
>>> sparse1 = SparseVector(2, [0, 1], [2., 1.])
>>> dense1.squared_distance(sparse1)
2.0
>>> dense1.squared_distance([1.,])
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
>>> dense1.squared_distance(SparseVector(1, [0,], [1.,]))
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
```

toArray() [source]

Returns the underlying numpy.ndarray

property **values**

Returns the underlying numpy.ndarray

class pyspark.ml.linalg.**SparseVector**(*size*, **args*) [source]

A simple sparse vector class for passing data to MLlib. Users may alternatively pass SciPy's {scipy.sparse} data types.

dot(*other*) [source]

Dot product with a SparseVector or 1- or 2-dimensional Numpy array.

```
>>> a = SparseVector(4, [1, 3], [3.0, 4.0])
>>> a.dot(a)
25.0
>>> a.dot(array.array('d', [1., 2., 3., 4.]))
22.0
>>> b = SparseVector(4, [2], [1.0])
>>> a.dot(b)
0.0
>>> a.dot(np.array([[1, 1], [2, 2], [3, 3], [4, 4]]))
array([ 22.,  22.])
>>> a.dot([1., 2., 3.])
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
>>> a.dot(np.array([1., 2.]))
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
>>> a.dot(DenseVector([1., 2.]))
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
>>> a.dot(np.zeros((3, 2)))
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
```

indices = *None*

A list of indices corresponding to active entries.

norm(*p*) [source]

Calculates the norm of a SparseVector.

```
>>> a = SparseVector(4, [0, 1], [3., -4.])
>>> a.norm(1)
7.0
>>> a.norm(2)
5.0
```

numNonzeros() [source]

Number of nonzero elements. This scans all active values and count non zeros.

size = *None*

Size of the vector.

squared_distance(*other*) [source]

Squared distance from a SparseVector or 1-dimensional NumPy array.

```
>>> a = SparseVector(4, [1, 3], [3.0, 4.0])
>>> a.squared_distance(a)
0.0
>>> a.squared_distance(array.array('d', [1., 2., 3., 4.]))
11.0
>>> a.squared_distance(np.array([1., 2., 3., 4.]))
11.0
>>> b = SparseVector(4, [2], [1.0])
>>> a.squared_distance(b)
26.0
>>> b.squared_distance(a)
26.0
>>> b.squared_distance([1., 2.])
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
>>> b.squared_distance(SparseVector(3, [1,], [1.0,]))
Traceback (most recent call last):
    ...
AssertionError: dimension mismatch
```

toArray() [source]

Returns a copy of this SparseVector as a 1-dimensional numpy.ndarray.

values = *None*

A list of values corresponding to active entries.

class pyspark.ml.linalg.**Vectors** [source]

Factory methods for working with vectors.

Note: Dense vectors are simply represented as NumPy array objects, so there is no need to covert them for use in MLlib. For sparse vectors, the factory methods in this class create an MLlib-compatible type, or users can pass in SciPy's *scipy.sparse* column vectors.

static **dense**(**elements*) [source]

Create a dense vector of 64-bit floats from a Python list or numbers.

```
>>> Vectors.dense([1, 2, 3])
DenseVector([1.0, 2.0, 3.0])
>>> Vectors.dense(1.0, 2.0)
DenseVector([1.0, 2.0])
```

static **norm**(*vector*, *p*) [source]

Find norm of the given vector.

static **sparse**(*size*, **args*) [source]

Create a sparse vector, using either a dictionary, a list of (index, value) pairs, or two separate arrays of indices and values (sorted by index).

Parameters:

- **size** – Size of the vector.
- **args** – Non-zero entries, as a dictionary, list of tuples, or two sorted lists containing indices and values.

```
>>> Vectors.sparse(4, {1: 1.0, 3: 5.5})
SparseVector(4, {1: 1.0, 3: 5.5})
>>> Vectors.sparse(4, [(1, 1.0), (3, 5.5)])
SparseVector(4, {1: 1.0, 3: 5.5})
>>> Vectors.sparse(4, [1, 3], [1.0, 5.5])
SparseVector(4, {1: 1.0, 3: 5.5})
```

static **squared_distance**(*v1*, *v2*) [source]

Squared distance between two vectors. a and b can be of type SparseVector, DenseVector, np.ndarray or array.array.

```
>>> a = Vectors.sparse(4, [(0, 1), (3, 4)])
>>> b = Vectors.dense([2, 5, 4, 1])
>>> a.squared_distance(b)
51.0
```

static **zeros**(*size*) [source]

class pyspark.ml.linalg.**Matrix**(*numRows*, *numCols*, *isTransposed=False*) [source]

toArray() [source]

Returns its elements in a numpy.ndarray.

class pyspark.ml.linalg.**DenseMatrix**(*numRows*, *numCols*, *values*, *isTransposed=False*) [source]

Column-major dense matrix.

toArray() [source]

Return a numpy.ndarray

```
>>> m = DenseMatrix(2, 2, range(4))
>>> m.toArray()
array([[ 0.,  2.],
       [ 1.,  3.]])
```

toSparse() [source]

Convert to SparseMatrix

class pyspark.ml.linalg.**SparseMatrix**(*numRows*, *numCols*, *colPtrs*, *rowIndices*, *values*, *isTransposed=False*) [source]

Sparse Matrix stored in CSC format.

toArray() [source]

Return a numpy.ndarray

toDense() [source]

class pyspark.ml.linalg.**Matrices** [source]

static **dense**(*numRows*, *numCols*, *values*) [source]

Create a DenseMatrix

static **sparse**(*numRows*, *numCols*, *colPtrs*, *rowIndices*, *values*) [source]

Create a SparseMatrix

pyspark.ml.recommendation module

class pyspark.ml.recommendation.**ALS**(*rank=10*, *maxIter=10*, *regParam=0.1*, *numUserBlocks=10*, *numItemBlocks=10*, *implicitPrefs=False*, *alpha=1.0*, *userCol='user'*, *itemCol='item'*, *seed=None*, *ratingCol='rating'*, *nonnegative=False*, *checkpointInterval=10*, *intermediateStorageLevel='MEMORY_AND_DISK'*, *finalStorageLevel='MEMORY_AND_DISK'*, *coldStartStrategy='nan'*) [source]

Alternating Least Squares (ALS) matrix factorization.

ALS attempts to estimate the ratings matrix *R* as the product of two lower-rank matrices, *X* and *Y*, i.e. *X * Yt = R*. Typically these approximations are called 'factor' matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix.

This is a blocked implementation of the ALS factorization algorithm that groups the two sets of factors (referred to as "users" and "products") into blocks and reduces communication by only sending one copy of each user vector to each product block on each iteration, and only for the product blocks that need that user's feature vector. This is achieved by pre-computing some information about the ratings matrix to determine the "out-links" of each user (which blocks of products it will contribute to) and "in-link" information for each product (which of the feature vectors it receives from each user block it will depend on). This allows us to send only an array of feature vectors between each user block and product block, and have the product block find the users' ratings and update the products based on these messages.

For implicit preference data, the algorithm used is based on "Collaborative Filtering for Implicit Feedback Datasets",, adapted for the blocked approach used here.

Essentially instead of finding the low-rank approximations to the rating matrix *R*, this finds the approximations for a preference matrix *P* where the elements of *P* are 1 if r > 0 and 0 if r <= 0. The ratings then act as 'confidence' values related to strength of indicated user preferences rather than explicit ratings given to items.

Note: the input rating dataframe to the ALS implementation should be deterministic. Nondeterministic data can cause failure during fitting ALS model. For example, an order-sensitive operation like sampling after a repartition makes dataframe output nondeterministic, like *df.repartition(2).sample(False, 0.5, 1618)*. Checkpointing sampled dataframe or adding a sort before sampling can help make the dataframe deterministic.

```
>>> df = spark.createDataFrame(
...     [(0, 0, 4.0), (0, 1, 2.0), (1, 1, 3.0), (1, 2, 4.0), (2, 1, 1.0), (2, 2, 5.0)],
...     ["user", "item", "rating"])
>>> als = ALS(rank=10, seed=0)
>>> als.setMaxIter(5)
ALS...
>>> als.getMaxIter()
5
>>> als.setRegParam(0.1)
ALS...
>>> als.getRegParam()
0.1
>>> als.clear(als.regParam)
>>> model = als.fit(df)
>>> model.getUserCol()
'user'
>>> model.setUserCol("user")
ALSModel...
>>> model.getItemCol()
'item'
>>> model.setPredictionCol("newPrediction")
ALS...
>>> model.rank
10
>>> model.userFactors.orderBy("id").collect()
[Row(id=0, features=[...]), Row(id=1, ...), Row(id=2, ...)]
>>> test = spark.createDataFrame([(0, 2), (1, 0), (2, 0)], ["user", "item"])
>>> predictions = sorted(model.transform(test).collect(), key=lambda r: r[0])
>>> predictions[0]
Row(user=0, item=2, newPrediction=0.6929101347923279)
>>> predictions[1]
Row(user=1, item=0, newPrediction=3.47356915473938)
>>> predictions[2]
Row(user=2, item=0, newPrediction=-0.8991986513137817)
>>> user_recs = model.recommendForAllUsers(3)
>>> user_recs.where(user_recs.user == 0)        .select("recommendations.item", "recommendations.
[Row(item=[0, 1, 2], rating=[3.910..., 1.997..., 0.692...])]
>>> item_recs = model.recommendForAllItems(3)
>>> item_recs.where(item_recs.item == 2)        .select("recommendations.user", "recommendations.
[Row(user=[2, 1, 0], rating=[4.892..., 3.991..., 0.692...])]
>>> user_subset = df.where(df.user == 2)
>>> user_subset_recs = model.recommendForUserSubset(user_subset, 3)
>>> user_subset_recs.select("recommendations.item", "recommendations.rating").first()
Row(item=[2, 1, 0], rating=[4.892..., 1.076..., -0.899...])
>>> item_subset = df.where(df.item == 0)
>>> item_subset_recs = model.recommendForItemSubset(item_subset, 3)
>>> item_subset_recs.select("recommendations.user", "recommendations.rating").first()
Row(user=[0, 1, 2], rating=[3.910..., 3.473..., -0.899...])
>>> als_path = temp_path + "/als"
>>> als.save(als_path)
>>> als2 = ALS.load(als_path)
>>> als.getMaxIter()
5
>>> model_path = temp_path + "/als_model"
>>> model.save(model_path)
>>> model2 = ALSModel.load(model_path)
>>> model.rank == model2.rank
True
>>> sorted(model.userFactors.collect()) == sorted(model2.userFactors.collect())
True
>>> sorted(model.itemFactors.collect()) == sorted(model2.itemFactors.collect())
True
```

New in version 1.4.0.

alpha = *Param(parent='undefined', name='alpha', doc='alpha for implicit preference')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or*

disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')

clear(*param*)

Clears a param from the param map if it has been explicitly set.

coldStartStrategy = *Param(parent='undefined', name='coldStartStrategy', doc="strategy for dealing with unknown or new users/items at prediction time. This may be useful in cross-validation or production scenarios, for handling user/item ids the model has not seen in the training data. Supported values: 'nan', 'drop'.")*

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

finalStorageLevel = *Param(parent='undefined', name='finalStorageLevel', doc='StorageLevel for ALS model factors.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getAlpha()

Gets the value of alpha or its default value.

New in version 1.4.0.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getColdStartStrategy()

Gets the value of coldStartStrategy or its default value.

New in version 2.2.0.

getFinalStorageLevel()

Gets the value of finalStorageLevel or its default value.

New in version 2.0.0.

getImplicitPrefs()

Gets the value of implicitPrefs or its default value.

New in version 1.4.0.

getIntermediateStorageLevel()

Gets the value of intermediateStorageLevel or its default value.

New in version 2.0.0.

getItemCol()

Gets the value of itemCol or its default value.

New in version 1.4.0.

getMaxIter()

Gets the value of maxIter or its default value.

getNonnegative()

Gets the value of nonnegative or its default value.

New in version 1.4.0.

getNumItemBlocks()
Gets the value of numItemBlocks or its default value.

New in version 1.4.0.

getNumUserBlocks()
Gets the value of numUserBlocks or its default value.

New in version 1.4.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getRank()
Gets the value of rank or its default value.

New in version 1.4.0.

getRatingCol()
Gets the value of ratingCol or its default value.

New in version 1.4.0.

getRegParam()
Gets the value of regParam or its default value.

getSeed()
Gets the value of seed or its default value.

getUserCol()
Gets the value of userCol or its default value.

New in version 1.4.0.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

implicitPrefs = *Param(parent='undefined', name='implicitPrefs', doc='whether to use implicit preference')*

intermediateStorageLevel = *Param(parent='undefined', name='intermediateStorageLevel', doc="StorageLevel for intermediate datasets. Cannot be 'NONE'.")*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

itemCol = *Param(parent='undefined', name='itemCol', doc='column name for item ids. Ids must be within the integer value range.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

nonnegative = *Param(parent='undefined', name='nonnegative', doc='whether to use nonnegative constraint for least squares')*

numItemBlocks = *Param(parent='undefined', name='numItemBlocks', doc='number of item blocks')*

numUserBlocks = *Param(parent='undefined', name='numUserBlocks', doc='number of user blocks')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

rank = *Param(parent='undefined', name='rank', doc='rank of the factorization')*

ratingCol = *Param(parent='undefined', name='ratingCol', doc='column name for ratings')*

classmethod **read**()
Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setAlpha(*value*) [source]
Sets the value of **alpha**.

New in version 1.4.0.

setCheckpointInterval(*value*) [source]
Sets the value of **checkpointInterval**.

setColdStartStrategy(*value*) [source]
Sets the value of **coldStartStrategy**.

New in version 2.2.0.

setFinalStorageLevel(*value*) [source]

Sets the value of **finalStorageLevel**.

New in version 2.0.0.

setImplicitPrefs(*value*) [source]

Sets the value of **implicitPrefs**.

New in version 1.4.0.

setIntermediateStorageLevel(*value*) [source]

Sets the value of **intermediateStorageLevel**.

New in version 2.0.0.

setItemCol(*value*) [source]

Sets the value of **itemCol**.

New in version 1.4.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

setNonnegative(*value*) [source]

Sets the value of **nonnegative**.

New in version 1.4.0.

setNumBlocks(*value*) [source]

Sets both **numUserBlocks** and **numItemBlocks** to the specific value.

New in version 1.4.0.

setNumItemBlocks(*value*) [source]

Sets the value of **numItemBlocks**.

New in version 1.4.0.

setNumUserBlocks(*value*) [source]

Sets the value of **numUserBlocks**.

New in version 1.4.0.

setParams(*self*, *rank=10*, *maxIter=10*, *regParam=0.1*, *numUserBlocks=10*, *numItemBlocks=10*, *implicitPrefs=False*, *alpha=1.0*, *userCol="user"*, *itemCol="item"*, *seed=None*, *ratingCol="rating"*, *nonnegative=False*, *checkpointInterval=10*, *intermediateStorageLevel="MEMORY_AND_DISK"*, *finalStorageLevel="MEMORY_AND_DISK"*, *coldStartStrategy="nan"*) [source]

Sets params for ALS.

New in version 1.4.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

setRank(*value*) [source]

Sets the value of **rank**.

New in version 1.4.0.

setRatingCol(*value*) [source]

Sets the value of **ratingCol**.

New in version 1.4.0.

setRegParam(*value*) [source]

Sets the value of **regParam**.

setSeed(*value*) [source]

Sets the value of **seed**.

setUserCol(*value*) [source]

Sets the value of **userCol**.

New in version 1.4.0.

userCol = *Param(parent='undefined', name='userCol', doc='column name for user ids. Ids must be within the integer value range.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.recommendation.**ALSModel**(*java_model=None*) [source]

Model fitted by ALS.

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

coldStartStrategy = *Param(parent='undefined', name='coldStartStrategy', doc="strategy for dealing with unknown or new users/items at prediction time. This may be useful in cross-validation or production scenarios, for handling user/item ids the model has not seen in the training data. Supported values: 'nan', 'drop'.")*

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getColdStartStrategy()

Gets the value of coldStartStrategy or its default value.

New in version 2.2.0.

getItemCol()

Gets the value of itemCol or its default value.

New in version 1.4.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getUserCol()

Gets the value of userCol or its default value.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

itemCol = *Param(parent='undefined', name='itemCol', doc='column name for item ids. Ids must be within the integer value range.')*

property **itemFactors**

a DataFrame that stores item factors in two columns: *id* and *features*

New in version 1.4.0.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

property **rank**

rank of the matrix factorization model

New in version 1.4.0.

classmethod **read**()

Returns an MLReader instance for this class.

recommendForAllItems(*numUsers*) [source]

Returns top *numUsers* users recommended for each item, for all items.

Parameters:
numUsers – max number of recommendations for each item
Returns:
a DataFrame of (itemCol, recommendations), where recommendations are stored as an array of (userCol, rating) Rows.

New in version 2.2.0.

recommendForAllUsers(*numItems*) [source]

Returns top *numItems* items recommended for each user, for all users.

Parameters:
numItems – max number of recommendations for each user
Returns:
a DataFrame of (userCol, recommendations), where recommendations are stored as an array of (itemCol, rating) Rows.

New in version 2.2.0.

recommendForItemSubset(*dataset*, *numUsers*) [source]

Returns top *numUsers* users recommended for each item id in the input data set. Note that if there are duplicate ids in the input dataset, only one set of recommendations per unique id will be returned.

Parameters:
- **dataset** – a Dataset containing a column of item ids. The column name must match *itemCol*.
- **numUsers** – max number of recommendations for each item

Returns:
a DataFrame of (itemCol, recommendations), where recommendations are stored as an array of (userCol, rating) Rows.

New in version 2.3.0.

recommendForUserSubset(*dataset*, *numItems*) [source]

Returns top *numItems* items recommended for each user id in the input data set. Note that if there are duplicate ids in the input dataset, only one set of recommendations per unique id will be returned.

Parameters:
- **dataset** – a Dataset containing a column of user ids. The column name must match *userCol*.
- **numItems** – max number of recommendations for each user

Returns:
a DataFrame of (userCol, recommendations), where recommendations are stored as an array of (itemCol, rating) Rows.

New in version 2.3.0.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setColdStartStrategy(*value*) [source]

Sets the value of **coldStartStrategy**.

New in version 3.0.0.

setItemCol(*value*) [source]

Sets the value of **itemCol**.

New in version 3.0.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 3.0.0.

setUserCol(*value*) [source]

Sets the value of **userCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

userCol = *Param(parent='undefined', name='userCol', doc='column name for user ids. Ids must be within the integer value range.')*

property **userFactors**

a DataFrame that stores user factors in two columns: *id* and *features*

New in version 1.4.0.

write()

Returns an MLWriter instance for this ML instance.

pyspark.ml.regression module

class pyspark.ml.regression.**AFTSurvivalRegression**(*featuresCol='features', labelCol='label', predictionCol='prediction', fitIntercept=True, maxIter=100, tol=1e-06, censorCol='censor', quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2*) [source]

Accelerated Failure Time (AFT) Model Survival Regression

Fit a parametric AFT survival regression model based on the Weibull distribution of the survival time.

See also: AFT Model

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0), 1.0),
...     (1e-40, Vectors.sparse(1, [], []), 0.0)], ["label", "features", "censor"])
>>> aftsr = AFTSurvivalRegression()
>>> aftsr.setMaxIter(10)
AFTSurvivalRegression...
>>> aftsr.getMaxIter()
10
>>> aftsr.clear(aftsr.maxIter)
>>> model = aftsr.fit(df)
>>> model.setFeaturesCol("features")
AFTSurvivalRegressionModel...
>>> model.predict(Vectors.dense(6.3))
1.0
>>> model.predictQuantiles(Vectors.dense(6.3))
DenseVector([0.0101, 0.0513, 0.1054, 0.2877, 0.6931, 1.3863, 2.3026, 2.9957, 4.6052])
>>> model.transform(df).show()
+-------+---------+------+----------+
|  label| features|censor|prediction|
+-------+---------+------+----------+
|    1.0|    [1.0]|   1.0|       1.0|
|1.0E-40|(1,[],[])|   0.0|       1.0|
+-------+---------+------+----------+
...
>>> aftsr_path = temp_path + "/aftsr"
>>> aftsr.save(aftsr_path)
>>> aftsr2 = AFTSurvivalRegression.load(aftsr_path)
>>> aftsr2.getMaxIter()
100
>>> model_path = temp_path + "/aftsr_model"
>>> model.save(model_path)
>>> model2 = AFTSurvivalRegressionModel.load(model_path)
>>> model.coefficients == model2.coefficients
True
>>> model.intercept == model2.intercept
True
>>> model.scale == model2.scale
True
```

New in version 1.6.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

censorCol = *Param(parent='undefined', name='censorCol', doc='censor column name. The value of this column could be 0 or 1. If the value is 1, it means the event has occurred i.e. uncensored; otherwise censored.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getAggregationDepth()

Gets the value of aggregationDepth or its default value.

getCensorCol()

Gets the value of censorCol or its default value.

New in version 1.6.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getFitIntercept()
Gets the value of fitIntercept or its default value.

getLabelCol()
Gets the value of labelCol or its default value.

getMaxIter()
Gets the value of maxIter or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getQuantileProbabilities()
Gets the value of quantileProbabilities or its default value.

New in version 1.6.0.

getQuantilesCol()
Gets the value of quantilesCol or its default value.

New in version 1.6.0.

getTol()
Gets the value of tol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

quantileProbabilities = *Param(parent='undefined', name='quantileProbabilities', doc='quantile probabilities array. Values of the quantile probabilities array should be in the range (0, 1) and the array should be non-empty.')*

quantilesCol = *Param(parent='undefined', name='quantilesCol', doc='quantiles column name. This column will output quantiles of corresponding quantileProbabilities if it is set.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setAggregationDepth(*value*) [source]
Sets the value of **aggregationDepth**.

New in version 2.1.0.

setCensorCol(*value*) [source]
Sets the value of **censorCol**.

New in version 1.6.0.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

New in version 1.6.0.

setFitIntercept(*value*) [source]
Sets the value of **fitIntercept**.

New in version 1.6.0.

setLabelCol(*value*) [source]
Sets the value of **labelCol**.

New in version 1.6.0.

setMaxIter(*value*) [source]
Sets the value of **maxIter**.

New in version 1.6.0.

setParams(*featuresCol='features', labelCol='label', predictionCol='prediction', fitIntercept=True, maxIter=100, tol=1e-06, censorCol='censor', quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2*) [source]

setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", fitIntercept=True, maxIter=100, tol=1E-6, censorCol="censor", quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2):

New in version 1.6.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 1.6.0.

setQuantileProbabilities(*value*) [source]

Sets the value of **quantileProbabilities**.

New in version 1.6.0.

setQuantilesCol(*value*) [source]

Sets the value of **quantilesCol**.

New in version 1.6.0.

setTol(*value*) [source]

Sets the value of **tol**.

New in version 1.6.0.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**AFTSurvivalRegressionModel**(*java_model=None*) [source]

Model fitted by **AFTSurvivalRegression**.

New in version 1.6.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

censorCol = *Param(parent='undefined', name='censorCol', doc='censor column name. The value of this column could be 0 or 1. If the value is 1, it means the event has occurred i.e. uncensored; otherwise censored.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

property **coefficients**

Model coefficients.

New in version 2.0.0.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

getAggregationDepth()

Gets the value of aggregationDepth or its default value.

getCensorCol()

Gets the value of censorCol or its default value.

New in version 1.6.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getFitIntercept()

Gets the value of fitIntercept or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getQuantileProbabilities()
Gets the value of quantileProbabilities or its default value.

New in version 1.6.0.

getQuantilesCol()
Gets the value of quantilesCol or its default value.

New in version 1.6.0.

getTol()
Gets the value of tol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

property **intercept**
Model intercept.

New in version 1.6.0.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*features*) [source]
Predicted value

New in version 2.0.0.

predictQuantiles(*features*) [source]
Predicted Quantiles

New in version 2.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

quantileProbabilities = *Param(parent='undefined', name='quantileProbabilities', doc='quantile probabilities array. Values of the quantile probabilities array should be in the range (0, 1) and the array should be non-empty.')*

quantilesCol = *Param(parent='undefined', name='quantilesCol', doc='quantiles column name. This column will output quantiles of corresponding quantileProbabilities if it is set.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

property **scale**
Model scale parameter.

New in version 1.6.0.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*) [source]
Sets the value of **predictionCol**.

New in version 3.0.0.

setQuantileProbabilities(*value*) [source]
Sets the value of **quantileProbabilities**.

New in version 3.0.0.

setQuantilesCol(*value*) [source]
Sets the value of **quantilesCol**.

New in version 3.0.0.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**DecisionTreeRegressor**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='variance', seed=None, varianceCol=None, weightCol=None, leafCol='', minWeightFractionPerNode=0.0*) [source]

Decision tree learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> dt = DecisionTreeRegressor(maxDepth=2)
>>> dt.setVarianceCol("variance")
DecisionTreeRegressor...
>>> model = dt.fit(df)
>>> model.getVarianceCol()
'variance'
>>> model.setLeafCol("leafId")
DecisionTreeRegressionModel...
>>> model.depth
1
>>> model.numNodes
3
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.predict(test0.head().features)
0.0
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> model.predictLeaf(test0.head().features)
0.0
>>> result.leafId
0.0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
>>> dtr_path = temp_path + "/dtr"
>>> dt.save(dtr_path)
>>> dt2 = DecisionTreeRegressor.load(dtr_path)
>>> dt2.getMaxDepth()
2
>>> model_path = temp_path + "/dtr_model"
>>> model.save(model_path)
>>> model2 = DecisionTreeRegressionModel.load(model_path)
>>> model.numNodes == model2.numNodes
True
>>> model.depth == model2.depth
True
>>> model.transform(test1).head().variance
0.0
```

```
>>> df3 = spark.createDataFrame([
...     (1.0, 0.2, Vectors.dense(1.0)),
...     (1.0, 0.8, Vectors.dense(1.0)),
...     (0.0, 1.0, Vectors.sparse(1, [], []))], ["label", "weight", "features"])
>>> dt3 = DecisionTreeRegressor(maxDepth=2, weightCol="weight", varianceCol="variance")
>>> model3 = dt3.fit(df3)
>>> print(model3.toDebugString)
DecisionTreeRegressionModel...depth=1, numNodes=3...
```

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:

extra – Extra parameters to copy to the new instance

Returns:

Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:

extra – extra param values

Returns:

merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of `pyspark.sql.DataFrame`
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of `pyspark.sql.DataFrame`.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.4.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()

Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()

Gets the value of minWeightFractionPerNode or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getVarianceCol()

Gets the value of varianceCol or its default value.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: variance')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setCacheNodeIds(*value*) [source]

Sets the value of **cacheNodeIds**.

New in version 1.4.0.

setCheckpointInterval(*value*) [source]

Sets the value of **checkpointInterval**.

New in version 1.4.0.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setImpurity(*value*) [source]

Sets the value of **impurity**.

New in version 1.4.0.

setLabelCol(*value*)

Sets the value of **labelCol**.

New in version 3.0.0.

setLeafCol(*value*)

Sets the value of **leafCol**.

setMaxBins(*value*) [source]

Sets the value of **maxBins**.

New in version 1.4.0.

setMaxDepth(*value*) [source]

Sets the value of **maxDepth**.

New in version 1.4.0.

setMaxMemoryInMB(*value*) [source]

Sets the value of **maxMemoryInMB**.

New in version 1.4.0.

setMinInfoGain(*value*) [source]

Sets the value of **minInfoGain**.

New in version 1.4.0.

setMinInstancesPerNode(*value*) [source]

Sets the value of **minInstancesPerNode**.

New in version 1.4.0.

setMinWeightFractionPerNode(*value*) [source]

Sets the value of **minWeightFractionPerNode**.

New in version 3.0.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity="variance"*, *seed=None*, *varianceCol=None*, *weightCol=None*, *leafCol=""*, *minWeightFractionPerNode=0.0*) [source]

Sets params for the DecisionTreeRegressor.

New in version 1.4.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

New in version 1.4.0.

setVarianceCol(*value*) [source]

Sets the value of **varianceCol**.

New in version 2.0.0.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

New in version 3.0.0.

supportedImpurities = *['variance']*

varianceCol = *Param(parent='undefined', name='varianceCol', doc='column name for the biased sample variance of prediction.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**DecisionTreeRegressionModel**(*java_model=None*) [source]

Model fitted by **DecisionTreeRegressor**.

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

property **depth**

Return depth of the decision tree.

New in version 1.5.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

property **featureImportances**

Estimate of the importance of each feature.

This generalizes the idea of “Gini” importance to other losses, following the explanation of Gini importance from “Random Forests” documentation by Leo Breiman and Adele Cutler, and following the implementation from scikit-learn.

This feature importance is calculated as follows:

- importance(feature j) = sum (over nodes which split on feature j) of the gain, where gain is scaled by the number of instances passing through node
- Normalize importances for tree to sum to 1.

Note: Feature importance for single decision trees can have high variance due to correlated predictor variables. Consider using a **RandomForestRegressor** to determine feature importance instead.

New in version 2.0.0.

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()
Gets the value of impurity or its default value.

New in version 1.4.0.

getLabelCol()
Gets the value of labelCol or its default value.

getLeafCol()
Gets the value of leafCol or its default value.

getMaxBins()
Gets the value of maxBins or its default value.

getMaxDepth()
Gets the value of maxDepth or its default value.

getMaxMemoryInMB()
Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()
Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()
Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()
Gets the value of minWeightFractionPerNode or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getSeed()
Gets the value of seed or its default value.

getVarianceCol()
Gets the value of varianceCol or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: variance')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **numFeatures**
Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **numNodes**
Return number of nodes of the decision tree.

New in version 1.5.0.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)

Predict label for the given features.

New in version 3.0.0.

predictLeaf(*value*)

Predict the indices of the leaves corresponding to the feature vector.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setLeafCol(*value*)

Sets the value of **leafCol**.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setVarianceCol(*value*) [source]

Sets the value of **varianceCol**.

New in version 3.0.0.

supportedImpurities = *['variance']*

property **toDebugString**

Full description of model.

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

varianceCol = *Param(parent='undefined', name='varianceCol', doc='column name for the biased sample variance of prediction.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**GBTRegressor**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, subsamplingRate=1.0, checkpointInterval=10, lossType='squared', maxIter=20, stepSize=0.1, seed=None, impurity='variance', featureSubsetStrategy='all', validationTol=0.01, validationIndicatorCol=None, leafCol='', minWeightFractionPerNode=0.0, weightCol=None*) [source]

Gradient-Boosted Trees (GBTs) learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> gbt = GBTRegressor(maxDepth=2, seed=42, leafCol="leafId")
>>> gbt.setMaxIter(5)
GBTRegressor...
>>> gbt.setMinWeightFractionPerNode(0.049)
GBTRegressor...
>>> gbt.getMaxIter()
5
>>> print(gbt.getImpurity())
variance
>>> print(gbt.getFeatureSubsetStrategy())
all
>>> model = gbt.fit(df)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> model.numFeatures
1
>>> allclose(model.treeWeights, [1.0, 0.1, 0.1, 0.1, 0.1])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.predict(test0.head().features)
0.0
>>> model.predictLeaf(test0.head().features)
DenseVector([0.0, 0.0, 0.0, 0.0, 0.0])
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> result.leafId
DenseVector([0.0, 0.0, 0.0, 0.0, 0.0])
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
>>> gbtr_path = temp_path + "gbtr"
>>> gbt.save(gbtr_path)
>>> gbt2 = GBTRegressor.load(gbtr_path)
>>> gbt2.getMaxDepth()
2
>>> model_path = temp_path + "gbtr_model"
>>> model.save(model_path)
>>> model2 = GBTRegressionModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
>>> model.treeWeights == model2.treeWeights
True
>>> model.trees
[DecisionTreeRegressionModel...depth=..., DecisionTreeRegressionModel...]
>>> validation = spark.createDataFrame([(0.0, Vectors.dense(-1.0))],
...              ["label", "features"])
>>> model.evaluateEachIteration(validation, "squared")
[0.0, 0.0, 0.0, 0.0, 0.0]
>>> gbt = gbt.setValidationIndicatorCol("validationIndicator")
>>> gbt.getValidationIndicatorCol()
'validationIndicator'
>>> gbt.getValidationTol()
0.01
```

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featureSubsetStrategy = *Param(parent='undefined', name='featureSubsetStrategy', doc="The number of features to consider for splits at each tree node. Supported options: 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(number of features)), 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features). default = 'auto'")*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:

A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeatureSubsetStrategy()

Gets the value of featureSubsetStrategy or its default value.

New in version 1.4.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.4.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getLossType()

Gets the value of lossType or its default value.

New in version 1.4.0.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxIter()

Gets the value of maxIter or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()

Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()

Gets the value of minWeightFractionPerNode or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getStepSize()

Gets the value of stepSize or its default value.

getSubsamplingRate()

Gets the value of subsamplingRate or its default value.

New in version 1.4.0.

getValidationIndicatorCol()

Gets the value of validationIndicatorCol or its default value.

getValidationTol()

Gets the value of validationTol or its default value.

New in version 3.0.0.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: variance')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

lossType = *Param(parent='undefined', name='lossType', doc='Loss function which GBT tries to minimize (case-insensitive). Supported options: squared, absolute')*

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setCacheNodeIds(*value*) [source]

Sets the value of **cacheNodeIds**.

New in version 1.4.0.

setCheckpointInterval(*value*) [source]

Sets the value of **checkpointInterval**.

New in version 1.4.0.

setFeatureSubsetStrategy(*value*) [source]

Sets the value of **featureSubsetStrategy**.

New in version 2.4.0.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setImpurity(*value*) [source]

Sets the value of **impurity**.

New in version 1.4.0.

setLabelCol(*value*)

Sets the value of **labelCol**.

New in version 3.0.0.

setLeafCol(*value*)

Sets the value of **leafCol**.

setLossType(*value*) [source]

Sets the value of **lossType**.

New in version 1.4.0.

setMaxBins(*value*) [source]

Sets the value of **maxBins**.

New in version 1.4.0.

setMaxDepth(*value*) [source]

Sets the value of **maxDepth**.

New in version 1.4.0.

setMaxIter(*value*) [source]

Sets the value of **maxIter**.

New in version 1.4.0.

setMaxMemoryInMB(*value*) [source]

Sets the value of **maxMemoryInMB**.

New in version 1.4.0.

setMinInfoGain(*value*) [source]

Sets the value of **minInfoGain**.

New in version 1.4.0.

setMinInstancesPerNode(*value*) [source]

Sets the value of **minInstancesPerNode**.

New in version 1.4.0.

setMinWeightFractionPerNode(*value*) [source]

Sets the value of **minWeightFractionPerNode**.

New in version 3.0.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *subsamplingRate=1.0*, *checkpointInterval=10*, *lossType="squared"*, *maxIter=20*, *stepSize=0.1*, *seed=None*, *impurity="variance"*, *featureSubsetStrategy="all"*, *validationTol=0.01*, *validationIndicatorCol=None*, *leafCol=""*, *minWeightFractionPerNode=0.0*, *weightCol=None*) [source]

Sets params for Gradient Boosted Tree Regression.

New in version 1.4.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

New in version 1.4.0.

setStepSize(*value*) [source]

Sets the value of **stepSize**.

New in version 1.4.0.

setSubsamplingRate(*value*) [source]

Sets the value of **subsamplingRate**.

New in version 1.4.0.

setValidationIndicatorCol(*value*) [source]

Sets the value of **validationIndicatorCol**.

New in version 3.0.0.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

New in version 3.0.0.

stepSize = *Param(parent='undefined', name='stepSize', doc='Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.')*

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedFeatureSubsetStrategies = *['auto', 'all', 'onethird', 'sqrt', 'log2']*

supportedImpurities = *['variance']*

supportedLossTypes = *['squared', 'absolute']*

validationIndicatorCol = *Param(parent='undefined', name='validationIndicatorCol', doc='name of the column that indicates whether each row is for training or for validation. False indicates training; true indicates validation.')*

validationTol = *Param(parent='undefined', name='validationTol', doc='Threshold for stopping early when fit with validation is used. If the error rate on the validation input changes by less than the validationTol, then learning will stop early (before `maxIter`). This parameter is ignored when fit without validation is used.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**GBTRegressionModel**(*java_model=None*) [source]

Model fitted by **GBTRegressor**.

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

evaluateEachIteration(*dataset*, *loss*) [source]

Method to compute error or loss for every iteration of gradient boosting.

Parameters:
- **dataset** – Test dataset to evaluate model on, where dataset is an instance of **pyspark.sql.DataFrame**
- **loss** – The loss function used to compute error. Supported options: squared, absolute

New in version 2.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

property **featureImportances**

Estimate of the importance of each feature.

Each feature's importance is the average of its importance across all trees in the ensemble The importance vector is normalized to sum to 1. This method is suggested by Hastie et al. (Hastie, Tibshirani, Friedman. "The Elements of Statistical Learning, 2nd Edition." 2001.) and follows the implementation from scikit-learn.

See also: **DecisionTreeRegressionModel.featureImportances**

New in version 2.0.0.

featureSubsetStrategy = *Param(parent='undefined', name='featureSubsetStrategy', doc="The number of features to consider for splits at each tree node. Supported options: 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(number of features)), 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features). default = 'auto'")*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeatureSubsetStrategy()

Gets the value of featureSubsetStrategy or its default value.

New in version 1.4.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.4.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getLossType()

Gets the value of lossType or its default value.

New in version 1.4.0.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxIter()

Gets the value of maxIter or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()

Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()

Gets the value of minWeightFractionPerNode or its default value.

property **getNumTrees**

Number of trees in ensemble.

New in version 2.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getStepSize()

Gets the value of stepSize or its default value.

getSubsamplingRate()

Gets the value of subsamplingRate or its default value.

New in version 1.4.0.

getValidationIndicatorCol()

Gets the value of validationIndicatorCol or its default value.

getValidationTol()

Gets the value of validationTol or its default value.

New in version 3.0.0.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: variance')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

lossType = *Param(parent='undefined', name='lossType', doc='Loss function which GBT tries to minimize (case-insensitive). Supported options: squared, absolute')*

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **numFeatures**

Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)

Predict label for the given features.

New in version 3.0.0.

predictLeaf(*value*)

Predict the indices of the leaves corresponding to the feature vector.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setLeafCol(*value*)
Sets the value of **leafCol**.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

stepSize = *Param(parent='undefined', name='stepSize', doc='Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator.')*

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedFeatureSubsetStrategies = *['auto', 'all', 'onethird', 'sqrt', 'log2']*

supportedImpurities = *['variance']*

supportedLossTypes = *['squared', 'absolute']*

property **toDebugString**
Full description of model.

New in version 2.0.0.

property **totalNumNodes**
Total number of nodes, summed over all trees in the ensemble.

New in version 2.0.0.

transform(*dataset*, *params=None*)
Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

property **treeWeights**
Return the weights for each tree

New in version 1.5.0.

property **trees**
Trees in this ensemble. Warning: These have null parent Estimators.

New in version 2.0.0.

validationIndicatorCol = *Param(parent='undefined', name='validationIndicatorCol', doc='name of the column that indicates whether each row is for training or for validation. False indicates training; true indicates validation.')*

validationTol = *Param(parent='undefined', name='validationTol', doc='Threshold for stopping early when fit with validation is used. If the error rate on the validation input changes by less than the validationTol, then learning will stop early (before `maxIter`). This parameter is ignored when fit without validation is used.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**GeneralizedLinearRegression**(*labelCol='label'*, *featuresCol='features'*, *predictionCol='prediction'*, *family='gaussian'*, *link=None*, *fitIntercept=True*, *maxIter=25*, *tol=1e-06*, *regParam=0.0*, *weightCol=None*, *solver='irls'*, *linkPredictionCol=None*, *variancePower=0.0*, *linkPower=None*, *offsetCol=None*, *aggregationDepth=2*) [source]
Generalized Linear Regression.

Fit a Generalized Linear Model specified by giving a symbolic description of the linear predictor (link function) and a description of the error distribution (family). It supports "gaussian", "binomial", "poisson", "gamma" and "tweedie" as family. Valid link functions for each family is listed below. The first link function of each family is the default one.

- "gaussian" -> "identity", "log", "inverse"
- "binomial" -> "logit", "probit", "cloglog"
- "poisson" -> "log", "identity", "sqrt"
- "gamma" -> "inverse", "identity", "log"
- "tweedie" -> power link function specified through "linkPower". The default link power in the tweedie family is 1 - variancePower.

See also: GLM

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(0.0, 0.0)),
...     (1.0, Vectors.dense(1.0, 2.0)),
...     (2.0, Vectors.dense(0.0, 0.0)),
...     (2.0, Vectors.dense(1.0, 1.0)),], ["label", "features"])
>>> glr = GeneralizedLinearRegression(family="gaussian", link="identity", linkPredictionCol="p")
>>> glr.setRegParam(0.1)
GeneralizedLinearRegression...
>>> glr.getRegParam()
0.1
>>> glr.clear(glr.regParam)
>>> glr.setMaxIter(10)
GeneralizedLinearRegression...
>>> glr.getMaxIter()
10
>>> glr.clear(glr.maxIter)
>>> model = glr.fit(df)
>>> model.setFeaturesCol("features")
GeneralizedLinearRegressionModel...
>>> model.getMaxIter()
25
>>> model.getAggregationDepth()
2
>>> transformed = model.transform(df)
>>> abs(transformed.head().prediction - 1.5) < 0.001
True
>>> abs(transformed.head().p - 1.5) < 0.001
True
>>> model.coefficients
DenseVector([1.5..., -1.0...])
>>> model.numFeatures
2
>>> abs(model.intercept - 1.5) < 0.001
True
>>> glr_path = temp_path + "/glr"
>>> glr.save(glr_path)
>>> glr2 = GeneralizedLinearRegression.load(glr_path)
>>> glr.getFamily() == glr2.getFamily()
True
>>> model_path = temp_path + "/glr_model"
>>> model.save(model_path)
>>> model2 = GeneralizedLinearRegressionModel.load(model_path)
>>> model.intercept == model2.intercept
True
>>> model.coefficients[0] == model2.coefficients[0]
True
```

New in version 2.0.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

family = *Param(parent='undefined', name='family', doc='The name of family which is a description of the error distribution to be used in the model. Supported options: gaussian (default), binomial, poisson, gamma and tweedie.')*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getAggregationDepth()
Gets the value of aggregationDepth or its default value.

getFamily()
Gets the value of family or its default value.

New in version 2.0.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getFitIntercept()
Gets the value of fitIntercept or its default value.

getLabelCol()
Gets the value of labelCol or its default value.

getLink()
Gets the value of link or its default value.

New in version 2.0.0.

getLinkPower()
Gets the value of linkPower or its default value.

New in version 2.2.0.

getLinkPredictionCol()
Gets the value of linkPredictionCol or its default value.

New in version 2.0.0.

getMaxIter()
Gets the value of maxIter or its default value.

getOffsetCol()
Gets the value of offsetCol or its default value.

New in version 2.3.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getRegParam()
Gets the value of regParam or its default value.

getSolver()
Gets the value of solver or its default value.

getTol()
Gets the value of tol or its default value.

getVariancePower()
Gets the value of variancePower or its default value.

New in version 2.2.0.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

link = *Param(parent='undefined', name='link', doc='The name of link function which provides the relationship between the linear predictor and the mean of the distribution function. Supported options: identity, log, inverse, logit, probit, cloglog and sqrt.')*

linkPower = *Param(parent='undefined', name='linkPower', doc='The index in the power link function. Only applicable to the Tweedie family.')*

linkPredictionCol = *Param(parent='undefined', name='linkPredictionCol', doc='link prediction (linear predictor) column name')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

offsetCol = *Param(parent='undefined', name='offsetCol', doc='The offset column name. If this is not set or empty, we treat all instance offsets as 0.0')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setAggregationDepth(*value*) [source]
Sets the value of **aggregationDepth**.

New in version 3.0.0.

setFamily(*value*) [source]
Sets the value of **family**.

New in version 2.0.0.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setFitIntercept(*value*) [source]
Sets the value of **fitIntercept**.

New in version 2.2.0.

setLabelCol(*value*)
Sets the value of **labelCol**.

New in version 3.0.0.

setLink(*value*) [source]
Sets the value of **link**.

New in version 2.0.0.

setLinkPower(*value*) [source]
Sets the value of **linkPower**.

New in version 2.2.0.

setLinkPredictionCol(*value*) [source]
Sets the value of **linkPredictionCol**.

New in version 2.0.0.

setMaxIter(*value*) [source]
Sets the value of **maxIter**.

New in version 2.0.0.

setOffsetCol(*value*) [source]
Sets the value of **offsetCol**.

New in version 2.3.0.

setParams(*self*, *labelCol="label"*, *featuresCol="features"*, *predictionCol="prediction"*, *family="gaussian"*, *link=None*, *fitIntercept=True*, *maxIter=25*, *tol=1e-6*, *regParam=0.0*, *weightCol=None*, *solver="irls"*, *linkPredictionCol=None*, *variancePower=0.0*, *linkPower=None*, *offsetCol=None*, *aggregationDepth=2*) [source]
Sets params for generalized linear regression.

New in version 2.0.0.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

setRegParam(*value*) [source]
Sets the value of **regParam**.

New in version 2.0.0.

setSolver(*value*) [source]
Sets the value of **solver**.

New in version 2.0.0.

setTol(*value*) [source]
Sets the value of **tol**.

New in version 2.0.0.

setVariancePower(*value*) [source]
Sets the value of **variancePower**.

New in version 2.2.0.

setWeightCol(*value*) [source]
Sets the value of **weightCol**.

New in version 2.0.0.

solver = *Param(parent='undefined', name='solver', doc='The solver algorithm for optimization. Supported options: irls.')*

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

variancePower = *Param(parent='undefined', name='variancePower', doc='The power in the variance function of the Tweedie distribution which characterizes the relationship between the variance and mean of the distribution. Only applicable for the Tweedie family. Supported values: 0 and [1, Inf).')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat*

all instance weights as 1.0.')

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**GeneralizedLinearRegressionModel**(*java_model=None*) [source]

Model fitted by **GeneralizedLinearRegression**.

New in version 2.0.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

property **coefficients**

Model coefficients.

New in version 2.0.0.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

evaluate(*dataset*) [source]

Evaluates the model on a test dataset.

Parameters:
dataset – Test dataset to evaluate model on, where dataset is an instance of **pyspark.sql.DataFrame**

New in version 2.0.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

family = *Param(parent='undefined', name='family', doc='The name of family which is a description of the error distribution to be used in the model. Supported options: gaussian (default), binomial, poisson, gamma and tweedie.')*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

getAggregationDepth()

Gets the value of aggregationDepth or its default value.

getFamily()

Gets the value of family or its default value.

New in version 2.0.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getFitIntercept()

Gets the value of fitIntercept or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getLink()

Gets the value of link or its default value.

New in version 2.0.0.

getLinkPower()

Gets the value of linkPower or its default value.

New in version 2.2.0.

getLinkPredictionCol()

Gets the value of linkPredictionCol or its default value.

New in version 2.0.0.

getMaxIter()

Gets the value of maxIter or its default value.

getOffsetCol()

Gets the value of offsetCol or its default value.

New in version 2.3.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getRegParam()
Gets the value of regParam or its default value.

getSolver()
Gets the value of solver or its default value.

getTol()
Gets the value of tol or its default value.

getVariancePower()
Gets the value of variancePower or its default value.

New in version 2.2.0.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

property **hasSummary**
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

property **intercept**
Model intercept.

New in version 2.0.0.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

link = *Param(parent='undefined', name='link', doc='The name of link function which provides the relationship between the linear predictor and the mean of the distribution function. Supported options: identity, log, inverse, logit, probit, cloglog and sqrt.')*

linkPower = *Param(parent='undefined', name='linkPower', doc='The index in the power link function. Only applicable to the Tweedie family.')*

linkPredictionCol = *Param(parent='undefined', name='linkPredictionCol', doc='link prediction (linear predictor) column name')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **numFeatures**
Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

offsetCol = *Param(parent='undefined', name='offsetCol', doc='The offset column name. If this is not set or empty, we treat all instance offsets as 0.0')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)
Predict label for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setLinkPredictionCol(*value*) [source]
Sets the value of **linkPredictionCol**.

New in version 3.0.0.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

solver = *Param(parent='undefined', name='solver', doc='The solver algorithm for optimization. Supported options: irls.')*

property **summary**

Gets summary (e.g. residuals, deviance, pValues) of model on training set. An exception is thrown if *trainingSummary is None*.

New in version 2.0.0.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

variancePower = *Param(parent='undefined', name='variancePower', doc='The power in the variance function of the Tweedie distribution which characterizes the relationship between the variance and mean of the distribution. Only applicable for the Tweedie family. Supported values: 0 and [1, Inf).')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**GeneralizedLinearRegressionSummary**(*java_obj=None*) [source]

Generalized linear regression results evaluated on a dataset.

New in version 2.0.0.

property **aic**

Akaike's "An Information Criterion"(AIC) for the fitted model.

New in version 2.0.0.

property **degreesOfFreedom**

Degrees of freedom.

New in version 2.0.0.

property **deviance**

The deviance for the fitted model.

New in version 2.0.0.

property **dispersion**

The dispersion of the fitted model. It is taken as 1.0 for the "binomial" and "poisson" families, and otherwise estimated by the residual Pearson's Chi-Squared statistic (which is defined as sum of the squares of the Pearson residuals) divided by the residual degrees of freedom.

New in version 2.0.0.

property **nullDeviance**

The deviance for the null model.

New in version 2.0.0.

property **numInstances**

Number of instances in DataFrame predictions.

New in version 2.2.0.

property **predictionCol**

Field in **predictions** which gives the predicted value of each instance. This is set to a new column name if the original model's *predictionCol* is not set.

New in version 2.0.0.

property **predictions**

Predictions output by the model's *transform* method.

New in version 2.0.0.

property **rank**

The numeric rank of the fitted linear model.

New in version 2.0.0.

property **residualDegreeOfFreedom**

The residual degrees of freedom.

New in version 2.0.0.

property **residualDegreeOfFreedomNull**

The residual degrees of freedom for the null model.

New in version 2.0.0.

residuals(*residualsType='deviance'*) [source]

Get the residuals of the fitted model by type.

Parameters:
residualsType – The type of residuals which should be returned. Supported options: deviance (default), pearson, working, and response.

New in version 2.0.0.

class pyspark.ml.regression.**GeneralizedLinearRegressionTrainingSummary**(*java_obj=None*) [source]

Generalized linear regression training results.

New in version 2.0.0.

property **aic**

Akaike's "An Information Criterion"(AIC) for the fitted model.

New in version 2.0.0.

property **coefficientStandardErrors**

Standard error of estimated coefficients and intercept.

If **GeneralizedLinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

New in version 2.0.0.

property **degreesOfFreedom**

Degrees of freedom.

New in version 2.0.0.

property **deviance**

The deviance for the fitted model.

New in version 2.0.0.

property **dispersion**

The dispersion of the fitted model. It is taken as 1.0 for the "binomial" and "poisson" families, and otherwise estimated by the residual Pearson's Chi-Squared statistic (which is defined as sum of the squares of the Pearson residuals) divided by the residual degrees of freedom.

New in version 2.0.0.

property **nullDeviance**

The deviance for the null model.

New in version 2.0.0.

property **numInstances**

Number of instances in DataFrame predictions.

New in version 2.2.0.

property **numIterations**

Number of training iterations.

New in version 2.0.0.

property **pValues**

Two-sided p-value of estimated coefficients and intercept.

If **GeneralizedLinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

New in version 2.0.0.

property **predictionCol**

Field in **predictions** which gives the predicted value of each instance. This is set to a new column name if the original model's *predictionCol* is not set.

New in version 2.0.0.

property **predictions**

Predictions output by the model's *transform* method.

New in version 2.0.0.

property **rank**

The numeric rank of the fitted linear model.

New in version 2.0.0.

property **residualDegreeOfFreedom**

The residual degrees of freedom.

New in version 2.0.0.

property **residualDegreeOfFreedomNull**

The residual degrees of freedom for the null model.

New in version 2.0.0.

residuals(*residualsType='deviance'*)

Get the residuals of the fitted model by type.

Parameters:
residualsType – The type of residuals which should be returned. Supported options: deviance (default), pearson, working, and response.

New in version 2.0.0.

property **solver**

The numeric solver used for training.

New in version 2.0.0.

property **tValues**

T-statistic of estimated coefficients and intercept.

If **GeneralizedLinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

New in version 2.0.0.

class pyspark.ml.regression.**IsotonicRegression**(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *weightCol=None*, *isotonic=True*, *featureIndex=0*) [source]

Currently implemented using parallelized pool adjacent violators algorithm. Only univariate (single feature) algorithm supported.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> ir = IsotonicRegression()
>>> model = ir.fit(df)
>>> model.setFeaturesCol("features")
IsotonicRegressionModel...
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> model.boundaries
DenseVector([0.0, 1.0])
>>> ir_path = temp_path + "/ir"
>>> ir.save(ir_path)
>>> ir2 = IsotonicRegression.load(ir_path)
>>> ir2.getIsotonic()
True
>>> model_path = temp_path + "/ir_model"
>>> model.save(model_path)
>>> model2 = IsotonicRegressionModel.load(model_path)
>>> model.boundaries == model2.boundaries
True
>>> model.predictions == model2.predictions
True
```

New in version 1.6.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featureIndex = *Param(parent='undefined', name='featureIndex', doc='The index of the feature if featuresCol is a vector column, no effect otherwise.')*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getFeatureIndex()

Gets the value of featureIndex or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getIsotonic()

Gets the value of isotonic or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

isotonic = *Param(parent='undefined', name='isotonic', doc='whether the output sequence should be isotonic/increasing (true) orantitonic/decreasing (false).')*

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setFeatureIndex(*value*) [source]
Sets the value of **featureIndex**.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

New in version 1.6.0.

setIsotonic(*value*) [source]
Sets the value of **isotonic**.

setLabelCol(*value*) [source]
Sets the value of **labelCol**.

New in version 1.6.0.

setParams(*featuresCol='features'*, *labelCol='label'*, *predictionCol='prediction'*, *weightCol=None*, *isotonic=True*, *featureIndex=0*) [source]
setParams(self, featuresCol="features", labelCol="label", predictionCol="prediction", weightCol=None, isotonic=True, featureIndex=0): Set the params for IsotonicRegression.

setPredictionCol(*value*) [source]
Sets the value of **predictionCol**.

New in version 1.6.0.

setWeightCol(*value*) [source]
Sets the value of **weightCol**.

New in version 1.6.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**IsotonicRegressionModel**(*java_model=None*) [source]
Model fitted by **IsotonicRegression**.

New in version 1.6.0.

property **boundaries**
Boundaries in increasing order for which predictions are known.

New in version 1.6.0.

clear(*param*)
Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)
Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featureIndex = *Param(parent='undefined', name='featureIndex', doc='The index of the feature if featuresCol is a vector column, no effect otherwise.')*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getFeatureIndex()

Gets the value of featureIndex or its default value.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getIsotonic()

Gets the value of isotonic or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

isotonic = *Param(parent='undefined', name='isotonic', doc='whether the output sequence should be isotonic/increasing (true) orantitonic/decreasing (false).')*

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

property **predictions**

Predictions associated with the boundaries at the same index, monotone because of isotonic regression.

New in version 1.6.0.

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeatureIndex(*value*) [source]

Sets the value of **featureIndex**.

setFeaturesCol(*value*) [source]

Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat*

all instance weights as 1.0.')

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**LinearRegression**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, standardization=True, solver='auto', weightCol=None, aggregationDepth=2, loss='squaredError', epsilon=1.35*) [source]

Linear regression.

The learning objective is to minimize the specified loss function, with regularization. This supports two kinds of loss:

- squaredError (a.k.a squared loss)
- huber (a hybrid of squared error for relatively small errors and absolute error for relatively large ones, and we estimate the scale parameter from training data)

This supports multiple types of regularization:

- none (a.k.a. ordinary least squares)
- L2 (ridge regression)
- L1 (Lasso)
- L2 + L1 (elastic net)

Note: Fitting with huber loss only supports none and L2 regularization.

```
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, 2.0, Vectors.dense(1.0)),
...     (0.0, 2.0, Vectors.sparse(1, [], []))], ["label", "weight", "features"])
>>> lr = LinearRegression(regParam=0.0, solver="normal", weightCol="weight")
>>> lr.setMaxIter(5)
LinearRegression...
>>> lr.getMaxIter()
5
>>> lr.setRegParam(0.1)
LinearRegression...
>>> lr.getRegParam()
0.1
>>> lr.setRegParam(0.0)
LinearRegression...
>>> model = lr.fit(df)
>>> model.setFeaturesCol("features")
LinearRegressionModel...
>>> model.setPredictionCol("newPrediction")
LinearRegressionModel...
>>> model.getMaxIter()
5
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> abs(model.predict(test0.head().features) - (-1.0)) < 0.001
True
>>> abs(model.transform(test0).head().newPrediction - (-1.0)) < 0.001
True
>>> abs(model.coefficients[0] - 1.0) < 0.001
True
>>> abs(model.intercept - 0.0) < 0.001
True
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> abs(model.transform(test1).head().newPrediction - 1.0) < 0.001
True
>>> lr.setParams("vector")
Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
>>> lr_path = temp_path + "/lr"
>>> lr.save(lr_path)
>>> lr2 = LinearRegression.load(lr_path)
>>> lr2.getMaxIter()
5
>>> model_path = temp_path + "/lr_model"
>>> model.save(model_path)
>>> model2 = LinearRegressionModel.load(model_path)
>>> model.coefficients[0] == model2.coefficients[0]
True
>>> model.intercept == model2.intercept
True
>>> model.numFeatures
1
>>> model.write().format("pmml").save(model_path + "_2")
```

New in version 1.4.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

elasticNetParam = *Param(parent='undefined', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.')*

epsilon = *Param(parent='undefined', name='epsilon', doc='The shape parameter to control the amount of robustness. Must be > 1.0. Only valid when loss is huber')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:

merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getAggregationDepth()

Gets the value of aggregationDepth or its default value.

getElasticNetParam()

Gets the value of elasticNetParam or its default value.

getEpsilon()

Gets the value of epsilon or its default value.

New in version 2.3.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getFitIntercept()

Gets the value of fitIntercept or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getLoss()

Gets the value of loss or its default value.

getMaxIter()

Gets the value of maxIter or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getRegParam()

Gets the value of regParam or its default value.

getSolver()

Gets the value of solver or its default value.

getStandardization()

Gets the value of standardization or its default value.

getTol()

Gets the value of tol or its default value.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

loss = *Param(parent='undefined', name='loss', doc='The loss function to be optimized. Supported options: squaredError, huber.')*

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setAggregationDepth(*value*) [source]
Sets the value of **aggregationDepth**.

setElasticNetParam(*value*) [source]
Sets the value of **elasticNetParam**.

setEpsilon(*value*) [source]
Sets the value of **epsilon**.

New in version 2.3.0.

setFeaturesCol(*value*)
Sets the value of **featuresCol**.

New in version 3.0.0.

setFitIntercept(*value*) [source]
Sets the value of **fitIntercept**.

setLabelCol(*value*)
Sets the value of **labelCol**.

New in version 3.0.0.

setLoss(*value*) [source]
Sets the value of **loss**.

setMaxIter(*value*) [source]
Sets the value of **maxIter**.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxIter=100*, *regParam=0.0*, *elasticNetParam=0.0*, *tol=1e-6*, *fitIntercept=True*, *standardization=True*, *solver="auto"*, *weightCol=None*, *aggregationDepth=2*, *loss="squaredError"*, *epsilon=1.35*) [source]
Sets params for linear regression.

New in version 1.4.0.

setPredictionCol(*value*)
Sets the value of **predictionCol**.

New in version 3.0.0.

setRegParam(*value*) [source]
Sets the value of **regParam**.

setSolver(*value*) [source]
Sets the value of **solver**.

setStandardization(*value*) [source]
Sets the value of **standardization**.

setTol(*value*) [source]
Sets the value of **tol**.

setWeightCol(*value*) [source]
Sets the value of **weightCol**.

solver = *Param(parent='undefined', name='solver', doc='The solver algorithm for optimization. Supported options: auto, normal, l-bfgs.')*

standardization = *Param(parent='undefined', name='standardization', doc='whether to standardize the training features before fitting the model.')*

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()
Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**LinearRegressionModel**(*java_model=None*) [source]
Model fitted by **LinearRegression**.

New in version 1.4.0.

aggregationDepth = *Param(parent='undefined', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2).')*

clear(*param*)
Clears a param from the param map if it has been explicitly set.

property **coefficients**
Model coefficients.

New in version 2.0.0.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

elasticNetParam = *Param(parent='undefined', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.')*

epsilon = *Param(parent='undefined', name='epsilon', doc='The shape parameter to control the amount of robustness. Must be > 1.0. Only valid when loss is huber')*

evaluate(*dataset*) [source]
Evaluates the model on a test dataset.

Parameters:
dataset – Test dataset to evaluate model on, where dataset is an instance of **pyspark.sql.DataFrame**

New in version 2.0.0.

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fitIntercept = *Param(parent='undefined', name='fitIntercept', doc='whether to fit an intercept term.')*

getAggregationDepth()
Gets the value of aggregationDepth or its default value.

getElasticNetParam()
Gets the value of elasticNetParam or its default value.

getEpsilon()
Gets the value of epsilon or its default value.

New in version 2.3.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getFitIntercept()
Gets the value of fitIntercept or its default value.

getLabelCol()
Gets the value of labelCol or its default value.

getLoss()
Gets the value of loss or its default value.

getMaxIter()
Gets the value of maxIter or its default value.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getRegParam()
Gets the value of regParam or its default value.

getSolver()
Gets the value of solver or its default value.

getStandardization()
Gets the value of standardization or its default value.

getTol()
Gets the value of tol or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

property **hasSummary**
Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

property **intercept**

Model intercept.

New in version 1.4.0.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

loss = *Param(parent='undefined', name='loss', doc='The loss function to be optimized. Supported options: squaredError, huber.')*

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

property **numFeatures**

Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)

Predict label for the given features.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

property **scale**

The value by which \(\|y - X'w\|\) is scaled down when loss is "huber", otherwise 1.0.

New in version 2.3.0.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

solver = *Param(parent='undefined', name='solver', doc='The solver algorithm for optimization. Supported options: auto, normal, l-bfgs.')*

standardization = *Param(parent='undefined', name='standardization', doc='whether to standardize the training features before fitting the model.')*

property **summary**

Gets summary (e.g. residuals, mse, r-squared) of model on training set. An exception is thrown if *trainingSummary is None.*

New in version 2.0.0.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0).')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an GeneralMLWriter instance for this ML instance.

class pyspark.ml.regression.**LinearRegressionSummary**(*java_obj=None*) [source]

Linear regression results evaluated on a dataset.

New in version 2.0.0.

property **coefficientStandardErrors**

Standard error of estimated coefficients and intercept. This value is only available when using the "normal" solver.

If **LinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

See also: **LinearRegression.solver**

New in version 2.0.0.

property **degreesOfFreedom**

Degrees of freedom.

New in version 2.2.0.

property **devianceResiduals**

The weighted residuals, the usual residuals rescaled by the square root of the instance weights.

New in version 2.0.0.

property **explainedVariance**

Returns the explained variance regression score. explainedVariance = \(1 - \frac{variance(y - \hat{y})}{variance(y)}\)

See also: Wikipedia explain variation

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **featuresCol**

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

property **labelCol**

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

property **meanAbsoluteError**

Returns the mean absolute error, which is a risk function corresponding to the expected value of the absolute error loss or l1-norm loss.

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **meanSquaredError**

Returns the mean squared error, which is a risk function corresponding to the expected value of the squared error loss or quadratic loss.

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **numInstances**

Number of instances in DataFrame predictions

New in version 2.0.0.

property **pValues**

Two-sided p-value of estimated coefficients and intercept. This value is only available when using the “normal” solver.

If **LinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

See also: **LinearRegression.solver**

New in version 2.0.0.

property **predictionCol**

Field in “predictions” which gives the predicted value of the label at each instance.

New in version 2.0.0.

property **predictions**

Dataframe outputted by the model’s *transform* method.

New in version 2.0.0.

property **r2**

Returns R^2, the coefficient of determination.

See also: Wikipedia coefficient of determination

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **r2adj**

Returns Adjusted R^2, the adjusted coefficient of determination.

See also: Wikipedia coefficient of determination, Adjusted R^2

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.4.0.

property **residuals**

Residuals (label - predicted value)

New in version 2.0.0.

property **rootMeanSquaredError**

Returns the root mean squared error, which is defined as the square root of the mean squared error.

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later

Spark versions.

New in version 2.0.0.

property **tValues**

T-statistic of estimated coefficients and intercept. This value is only available when using the “normal” solver.

If **LinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

See also: **LinearRegression.solver**

New in version 2.0.0.

class pyspark.ml.regression.**LinearRegressionTrainingSummary**(*java_obj=None*) [source]

Linear regression training results. Currently, the training summary ignores the training weights except for the objective trace.

New in version 2.0.0.

property **coefficientStandardErrors**

Standard error of estimated coefficients and intercept. This value is only available when using the “normal” solver.

If **LinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

See also: **LinearRegression.solver**

New in version 2.0.0.

property **degreesOfFreedom**

Degrees of freedom.

New in version 2.2.0.

property **devianceResiduals**

The weighted residuals, the usual residuals rescaled by the square root of the instance weights.

New in version 2.0.0.

property **explainedVariance**

Returns the explained variance regression score. explainedVariance = $1 - \frac{variance(y - \hat{y})}{variance(y)}$

See also: Wikipedia explain variation

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **featuresCol**

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

property **labelCol**

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

property **meanAbsoluteError**

Returns the mean absolute error, which is a risk function corresponding to the expected value of the absolute error loss or l1-norm loss.

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **meanSquaredError**

Returns the mean squared error, which is a risk function corresponding to the expected value of the squared error loss or quadratic loss.

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **numInstances**

Number of instances in DataFrame predictions

New in version 2.0.0.

property **objectiveHistory**

Objective function (scaled loss + regularization) at each iteration. This value is only available when using the “l-bfgs” solver.

See also: **LinearRegression.solver**

New in version 2.0.0.

property **pValues**

Two-sided p-value of estimated coefficients and intercept. This value is only available when using the “normal” solver.

If **LinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

See also: **LinearRegression.solver**

New in version 2.0.0.

property **predictionCol**

Field in “predictions” which gives the predicted value of the label at each instance.

New in version 2.0.0.

property **predictions**

Dataframe outputted by the model's *transform* method.

New in version 2.0.0.

property **r2**

Returns R^2, the coefficient of determination.

See also: Wikipedia coefficient of determination

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **r2adj**

Returns Adjusted R^2, the adjusted coefficient of determination.

See also: Wikipedia coefficient of determination, Adjusted R^2

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.4.0.

property **residuals**

Residuals (label - predicted value)

New in version 2.0.0.

property **rootMeanSquaredError**

Returns the root mean squared error, which is defined as the square root of the mean squared error.

Note: This ignores instance weights (setting all to 1.0) from *LinearRegression.weightCol*. This will change in later Spark versions.

New in version 2.0.0.

property **tValues**

T-statistic of estimated coefficients and intercept. This value is only available when using the "normal" solver.

If **LinearRegression.fitIntercept** is set to True, then the last element returned corresponds to the intercept.

See also: **LinearRegression.solver**

New in version 2.0.0.

property **totalIterations**

Number of training iterations until termination. This value is only available when using the "l-bfgs" solver.

See also: **LinearRegression.solver**

New in version 2.0.0.

class pyspark.ml.regression.**RandomForestRegressor**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='variance', subsamplingRate=1.0, seed=None, numTrees=20, featureSubsetStrategy='auto', leafCol='', minWeightFractionPerNode=0.0*) [source]

Random Forest learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from numpy import allclose
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> rf = RandomForestRegressor(numTrees=2, maxDepth=2)
>>> rf.setSeed(42)
RandomForestRegressor...
>>> model = rf.fit(df)
>>> model.getSeed()
42
>>> model.setLeafCol("leafId")
RandomForestRegressionModel...
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 1.0])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.predict(test0.head().features)
0.0
>>> model.predictLeaf(test0.head().features)
DenseVector([0.0, 0.0])
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> result.leafId
DenseVector([0.0, 0.0])
>>> model.numFeatures
1
>>> model.trees
[DecisionTreeRegressionModel...depth=..., DecisionTreeRegressionModel...]
>>> model.getNumTrees
2
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
0.5
>>> rfr_path = temp_path + "/rfr"
>>> rf.save(rfr_path)
>>> rf2 = RandomForestRegressor.load(rfr_path)
>>> rf2.getNumTrees()
2
>>> model_path = temp_path + "/rfr_model"
>>> model.save(model_path)
>>> model2 = RandomForestRegressionModel.load(model_path)
>>> model.featureImportances == model2.featureImportances
True
```

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or*

disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

featureSubsetStrategy = *Param(parent='undefined', name='featureSubsetStrategy', doc="The number of features to consider for splits at each tree node. Supported options: 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(number of features)), 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features). default = 'auto'")*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

Note: DeveloperApi

New in version 2.3.0.

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeatureSubsetStrategy()

Gets the value of featureSubsetStrategy or its default value.

New in version 1.4.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.4.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()
Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()
Gets the value of minWeightFractionPerNode or its default value.

getNumTrees()
Gets the value of numTrees or its default value.

New in version 1.4.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getSeed()
Gets the value of seed or its default value.

getSubsamplingRate()
Gets the value of subsamplingRate or its default value.

New in version 1.4.0.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: variance')*

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

numTrees = *Param(parent='undefined', name='numTrees', doc='Number of trees to train (>= 1).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setCacheNodeIds(*value*) [source]
Sets the value of **cacheNodeIds**.

setCheckpointInterval(*value*) [source]
Sets the value of **checkpointInterval**.

setFeatureSubsetStrategy(*value*) [source]
Sets the value of **featureSubsetStrategy**.

New in version 2.4.0.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setImpurity(*value*) [source]

Sets the value of **impurity**.

New in version 1.4.0.

setLabelCol(*value*)

Sets the value of **labelCol**.

New in version 3.0.0.

setLeafCol(*value*)

Sets the value of **leafCol**.

setMaxBins(*value*) [source]

Sets the value of **maxBins**.

setMaxDepth(*value*) [source]

Sets the value of **maxDepth**.

setMaxMemoryInMB(*value*) [source]

Sets the value of **maxMemoryInMB**.

setMinInfoGain(*value*) [source]

Sets the value of **minInfoGain**.

setMinInstancesPerNode(*value*) [source]

Sets the value of **minInstancesPerNode**.

setNumTrees(*value*) [source]

Sets the value of **numTrees**.

New in version 1.4.0.

setParams(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxDepth=5*, *maxBins=32*, *minInstancesPerNode=1*, *minInfoGain=0.0*, *maxMemoryInMB=256*, *cacheNodeIds=False*, *checkpointInterval=10*, *impurity="variance"*, *subsamplingRate=1.0*, *seed=None*, *numTrees=20*, *featureSubsetStrategy="auto"*, *leafCol=""*, *minWeightFractionPerNode=0.0*) [source]

Sets params for linear regression.

New in version 1.4.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

setSubsamplingRate(*value*) [source]

Sets the value of **subsamplingRate**.

New in version 1.4.0.

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedFeatureSubsetStrategies = *['auto', 'all', 'onethird', 'sqrt', 'log2']*

supportedImpurities = *['variance']*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.regression.**RandomForestRegressionModel**(*java_model=None*) [source]

Model fitted by **RandomForestRegressor**.

New in version 1.4.0.

cacheNodeIds = *Param(parent='undefined', name='cacheNodeIds', doc='If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algorithm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often should the cache be checkpointed or disable it by setting checkpointInterval.')*

checkpointInterval = *Param(parent='undefined', name='checkpointInterval', doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means that the cache will get checkpointed every 10 iterations. Note: this setting will be ignored if the checkpoint directory is not set in the SparkContext.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

property **featureImportances**

Estimate of the importance of each feature.

Each feature's importance is the average of its importance across all trees in the ensemble The importance vector is normalized to sum to 1. This method is suggested by Hastie et al. (Hastie, Tibshirani, Friedman. "The Elements of Statistical Learning, 2nd Edition." 2001.) and follows the implementation from scikit-learn.

See also: **DecisionTreeRegressionModel.featureImportances**

New in version 2.0.0.

featureSubsetStrategy = *Param(parent='undefined', name='featureSubsetStrategy', doc="The number of features to consider for splits at each tree node. Supported options: 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression), 'all' (use all features), 'onethird' (use 1/3 of the features), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(number of features)), 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features). default = 'auto'")*

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getCacheNodeIds()

Gets the value of cacheNodeIds or its default value.

getCheckpointInterval()

Gets the value of checkpointInterval or its default value.

getFeatureSubsetStrategy()

Gets the value of featureSubsetStrategy or its default value.

New in version 1.4.0.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getImpurity()

Gets the value of impurity or its default value.

New in version 1.4.0.

getLabelCol()

Gets the value of labelCol or its default value.

getLeafCol()

Gets the value of leafCol or its default value.

getMaxBins()

Gets the value of maxBins or its default value.

getMaxDepth()

Gets the value of maxDepth or its default value.

getMaxMemoryInMB()

Gets the value of maxMemoryInMB or its default value.

getMinInfoGain()

Gets the value of minInfoGain or its default value.

getMinInstancesPerNode()

Gets the value of minInstancesPerNode or its default value.

getMinWeightFractionPerNode()

Gets the value of minWeightFractionPerNode or its default value.

property **getNumTrees**

Number of trees in ensemble.

New in version 2.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getSeed()

Gets the value of seed or its default value.

getSubsamplingRate()

Gets the value of subsamplingRate or its default value.

New in version 1.4.0.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

impurity = *Param(parent='undefined', name='impurity', doc='Criterion used for information gain calculation (case-insensitive). Supported options: variance')*

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

leafCol = *Param(parent='undefined', name='leafCol', doc='Leaf indices column name. Predicted leaf index of each instance in each tree by preorder.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

maxBins = *Param(parent='undefined', name='maxBins', doc='Max number of bins for discretizing continuous features. Must be >=2 and >= number of categories for any categorical feature.')*

maxDepth = *Param(parent='undefined', name='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.')*

maxMemoryInMB = *Param(parent='undefined', name='maxMemoryInMB', doc='Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per iteration, and its aggregates may exceed this size.')*

minInfoGain = *Param(parent='undefined', name='minInfoGain', doc='Minimum information gain for a split to be considered at a tree node.')*

minInstancesPerNode = *Param(parent='undefined', name='minInstancesPerNode', doc='Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1.')*

minWeightFractionPerNode = *Param(parent='undefined', name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample count that each child must have after split. If a split causes the fraction of the total weight in the left or right child to be less than minWeightFractionPerNode, the split will be discarded as invalid. Should be in interval [0.0, 0.5).')*

property **numFeatures**

Returns the number of features the model was trained on. If unknown, returns -1

New in version 2.1.0.

numTrees = *Param(parent='undefined', name='numTrees', doc='Number of trees to train (>= 1).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predict(*value*)

Predict label for the given features.

New in version 3.0.0.

predictLeaf(*value*)

Predict the indices of the leaves corresponding to the feature vector.

New in version 3.0.0.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

New in version 3.0.0.

setLeafCol(*value*)

Sets the value of **leafCol**.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

New in version 3.0.0.

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedFeatureSubsetStrategies = *['auto', 'all', 'onethird', 'sqrt', 'log2']*

supportedImpurities = *['variance']*

property **toDebugString**

Full description of model.

New in version 2.0.0.

property **totalNumNodes**

Total number of nodes, summed over all trees in the ensemble.

New in version 2.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**

- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

property **treeWeights**

Return the weights for each tree

New in version 1.5.0.

property **trees**

Trees in this ensemble. Warning: These have null parent Estimators.

New in version 2.0.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

pyspark.ml.stat module

class pyspark.ml.stat.**ChiSquareTest** [source]

Conduct Pearson's independence test for every feature against the label. For each feature, the (feature, label) pairs are converted into a contingency matrix for which the Chi-squared statistic is computed. All label and feature values must be categorical.

The null hypothesis is that the occurrence of the outcomes is statistically independent.

New in version 2.2.0.

static **test**(*dataset*, *featuresCol*, *labelCol*) [source]

Perform a Pearson's independence test using dataset.

Parameters:
- **dataset** – DataFrame of categorical labels and categorical features. Real-valued features will be treated as categorical for each distinct value.
- **featuresCol** – Name of features column in dataset, of type *Vector* (*VectorUDT*).
- **labelCol** – Name of label column in dataset, of any numerical type.

Returns:
DataFrame containing the test result for every feature against the label. This DataFrame will contain a single Row with the following fields: - *pValues: Vector* - *degreesOfFreedom: Array[Int]* - *statistics: Vector* Each of these fields has one value per feature.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.stat import ChiSquareTest
>>> dataset = [[0, Vectors.dense([0, 0, 1])],
...            [0, Vectors.dense([1, 0, 1])],
...            [1, Vectors.dense([2, 1, 1])],
...            [1, Vectors.dense([3, 1, 1])]]
>>> dataset = spark.createDataFrame(dataset, ["label", "features"])
>>> chiSqResult = ChiSquareTest.test(dataset, 'features', 'label')
>>> chiSqResult.select("degreesOfFreedom").collect()[0]
Row(degreesOfFreedom=[3, 1, 0])
```

New in version 2.2.0.

class pyspark.ml.stat.**Correlation** [source]

Compute the correlation matrix for the input dataset of Vectors using the specified method. Methods currently supported: *pearson* (default), *spearman*.

Note: For Spearman, a rank correlation, we need to create an RDD[Double] for each column and sort it in order to retrieve the ranks and then join the columns back into an RDD[Vector], which is fairly costly. Cache the input Dataset before calling corr with *method = 'spearman'* to avoid recomputing the common lineage.

New in version 2.2.0.

static **corr**(*dataset*, *column*, *method='pearson'*) [source]

Compute the correlation matrix with specified method using dataset.

Parameters:
- **dataset** – A Dataset or a DataFrame.
- **column** – The name of the column of vectors for which the correlation coefficient needs to be computed. This must be a column of the dataset, and it must contain Vector objects.
- **method** – String specifying the method to use for computing correlation. Supported: *pearson* (default), *spearman*.

Returns:
A DataFrame that contains the correlation matrix of the column of vectors. This DataFrame contains a single row and a single column of name '$METHODNAME($COLUMN)'.

```
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.stat import Correlation
>>> dataset = [[Vectors.dense([1, 0, 0, -2])],
...            [Vectors.dense([4, 5, 0, 3])],
...            [Vectors.dense([6, 7, 0, 8])],
...            [Vectors.dense([9, 0, 0, 1])]]
>>> dataset = spark.createDataFrame(dataset, ['features'])
>>> pearsonCorr = Correlation.corr(dataset, 'features', 'pearson').collect()[0][0]
>>> print(str(pearsonCorr).replace('nan', 'NaN'))
DenseMatrix([[ 1.        ,  0.0556...,         NaN,  0.4004...],
             [ 0.0556...,  1.        ,         NaN,  0.9135...],
             [        NaN,         NaN,  1.        ,         NaN],
             [ 0.4004...,  0.9135...,         NaN,  1.        ]])
>>> spearmanCorr = Correlation.corr(dataset, 'features', method='spearman').collect()[0][0]
>>> print(str(spearmanCorr).replace('nan', 'NaN'))
DenseMatrix([[ 1.        ,  0.1054...,         NaN,  0.4       ],
             [ 0.1054...,  1.        ,         NaN,  0.9486... ],
             [        NaN,         NaN,  1.        ,         NaN],
             [ 0.4       ,  0.9486... ,         NaN,  1.        ]])
```

New in version 2.2.0.

class pyspark.ml.stat.**KolmogorovSmirnovTest** [source]

Conduct the two-sided Kolmogorov Smirnov (KS) test for data sampled from a continuous distribution.

By comparing the largest difference between the empirical cumulative distribution of the sample data and the theoretical distribution we can provide a test for the the null hypothesis that the sample data comes from that theoretical distribution.

New in version 2.4.0.

static **test**(*dataset*, *sampleCol*, *distName*, **params*) [source]

Conduct a one-sample, two-sided Kolmogorov-Smirnov test for probability distribution equality. Currently supports the normal distribution, taking as parameters the mean and standard deviation.

Parameters:

- **dataset** – a Dataset or a DataFrame containing the sample of data to test.
- **sampleCol** – Name of sample column in dataset, of any numerical type.
- **distName** – a *string* name for a theoretical distribution, currently only support "norm".
- **params** – a list of *Double* values specifying the parameters to be used for the theoretical distribution. For "norm" distribution, the parameters includes mean and variance.

Returns:

A DataFrame that contains the Kolmogorov-Smirnov test result for the input sampled data. This DataFrame will contain a single Row with the following fields: - *pValue: Double* - *statistic: Double*

```
>>> from pyspark.ml.stat import KolmogorovSmirnovTest
>>> dataset = [[-1.0], [0.0], [1.0]]
>>> dataset = spark.createDataFrame(dataset, ['sample'])
>>> ksResult = KolmogorovSmirnovTest.test(dataset, 'sample', 'norm', 0.0, 1.0).first()
>>> round(ksResult.pValue, 3)
1.0
>>> round(ksResult.statistic, 3)
0.175
>>> dataset = [[2.0], [3.0], [4.0]]
>>> dataset = spark.createDataFrame(dataset, ['sample'])
>>> ksResult = KolmogorovSmirnovTest.test(dataset, 'sample', 'norm', 3.0, 1.0).first()
>>> round(ksResult.pValue, 3)
1.0
>>> round(ksResult.statistic, 3)
0.175
```

New in version 2.4.0.

class pyspark.ml.stat.**Summarizer** [source]

Tools for vectorized statistics on MLlib Vectors. The methods in this package provide various statistics for Vectors contained inside DataFrames. This class lets users pick the statistics they would like to extract for a given column.

```
>>> from pyspark.ml.stat import Summarizer
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> summarizer = Summarizer.metrics("mean", "count")
>>> df = sc.parallelize([Row(weight=1.0, features=Vectors.dense(1.0, 1.0, 1.0)),
...                      Row(weight=0.0, features=Vectors.dense(1.0, 2.0, 3.0))]).toDF()
>>> df.select(summarizer.summary(df.features, df.weight)).show(truncate=False)
+-----------------------------------+
|aggregate_metrics(features, weight)|
+-----------------------------------+
|[[1.0,1.0,1.0], 1]                 |
+-----------------------------------+

>>> df.select(summarizer.summary(df.features)).show(truncate=False)
+---------------------------------+
|aggregate_metrics(features, 1.0)|
+---------------------------------+
|[[1.0,1.5,2.0], 2]               |
+---------------------------------+

>>> df.select(Summarizer.mean(df.features, df.weight)).show(truncate=False)
+--------------+
|mean(features)|
+--------------+
|[1.0,1.0,1.0] |
+--------------+

>>> df.select(Summarizer.mean(df.features)).show(truncate=False)
+--------------+
|mean(features)|
+--------------+
|[1.0,1.5,2.0] |
+--------------+
```

New in version 2.4.0.

static **count**(*col*, *weightCol=None*) [source]

return a column of count summary

New in version 2.4.0.

static **max**(*col*, *weightCol=None*) [source]

return a column of max summary

New in version 2.4.0.

static **mean**(*col*, *weightCol=None*) [source]

return a column of mean summary

New in version 2.4.0.

static **metrics**(**metrics*) [source]

Given a list of metrics, provides a builder that it turns computes metrics from a column.

See the documentation of [[Summarizer]] for an example.

The following metrics are accepted (case sensitive):

- mean: a vector that contains the coefficient-wise mean.
- sum: a vector that contains the coefficient-wise sum.
- variance: a vector tha contains the coefficient-wise variance.
- std: a vector tha contains the coefficient-wise standard deviation.
- count: the count of all vectors seen.
- numNonzeros: a vector with the number of non-zeros for each coefficients
- max: the maximum for each coefficient.
- min: the minimum for each coefficient.
- normL2: the Euclidean norm for each coefficient.
- normL1: the L1 norm of each coefficient (sum of the absolute values).

Parameters:

metrics – metrics that can be provided.

Returns:

an object of **pyspark.ml.stat.SummaryBuilder**

Note: Currently, the performance of this interface is about 2x~3x slower then using the RDD interface.

New in version 2.4.0.

static **min**(*col*, *weightCol=None*) [source]

return a column of min summary

New in version 2.4.0.

static **normL1**(*col*, *weightCol=None*) [source]

return a column of normL1 summary

New in version 2.4.0.

static **normL2**(*col*, *weightCol=None*) [source]

return a column of normL2 summary

New in version 2.4.0.

static **numNonZeros**(*col*, *weightCol=None*) [source]

return a column of numNonZero summary

New in version 2.4.0.

static **std**(*col*, *weightCol=None*) [source]

return a column of std summary

New in version 3.0.0.

static **sum**(*col*, *weightCol=None*) [source]

return a column of sum summary

New in version 3.0.0.

static **variance**(*col*, *weightCol=None*) [source]

return a column of variance summary

New in version 2.4.0.

class pyspark.ml.stat.**SummaryBuilder**(*jSummaryBuilder*) [source]

A builder object that provides summary statistics about a given column.

Users should not directly create such builders, but instead use one of the methods in **pyspark.ml.stat.Summarizer**

New in version 2.4.0.

summary(*featuresCol*, *weightCol=None*) [source]

Returns an aggregate object that contains the summary of the column with the requested metrics.

Parameters:

- **featuresCol** – a column that contains features Vector object.
- **weightCol** – a column that contains weight value. Default weight is 1.0.

Returns:

an aggregate column that contains the statistics. The exact content of this structure is determined during the creation of the builder.

New in version 2.4.0.

pyspark.ml.tuning module

class pyspark.ml.tuning.**ParamGridBuilder** [source]

Builder for a param grid used in grid search-based model selection.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> lr = LogisticRegression()
>>> output = ParamGridBuilder() \
...     .baseOn({lr.labelCol: 'l'}) \
...     .baseOn([lr.predictionCol, 'p']) \
...     .addGrid(lr.regParam, [1.0, 2.0]) \
...     .addGrid(lr.maxIter, [1, 5]) \
...     .build()
>>> expected = [
...     {lr.regParam: 1.0, lr.maxIter: 1, lr.labelCol: 'l', lr.predictionCol: 'p'},
...     {lr.regParam: 2.0, lr.maxIter: 1, lr.labelCol: 'l', lr.predictionCol: 'p'},
...     {lr.regParam: 1.0, lr.maxIter: 5, lr.labelCol: 'l', lr.predictionCol: 'p'},
...     {lr.regParam: 2.0, lr.maxIter: 5, lr.labelCol: 'l', lr.predictionCol: 'p'}]
>>> len(output) == len(expected)
True
>>> all([m in expected for m in output])
True
```

New in version 1.4.0.

addGrid(*param*, *values*) [source]

Sets the given parameters in this grid to fixed values.

param must be an instance of Param associated with an instance of Params (such as Estimator or Transformer).

New in version 1.4.0.

baseOn(**args*) [source]

Sets the given parameters in this grid to fixed values. Accepts either a parameter dictionary or a list of (parameter, value) pairs.

New in version 1.4.0.

build() [source]

Builds and returns all combinations of parameters specified by the param grid.

New in version 1.4.0.

class pyspark.ml.tuning.**CrossValidator**(*estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *numFolds=3*, *seed=None*, *parallelism=1*, *collectSubModels=False*) [source]

K-fold cross validation performs model selection by splitting the dataset into a set of non-overlapping randomly partitioned folds which are used as separate training and test datasets e.g., with k=3 folds, K-fold cross validation will generate 3 (training, test) dataset pairs, each of which uses 2/3 of the data for training and 1/3 for testing. Each fold is used as the test set exactly once.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.tuning import CrossValidatorModel
>>> import tempfile
>>> dataset = spark.createDataFrame(
...     [(Vectors.dense([0.0]), 0.0),
...      (Vectors.dense([0.4]), 1.0),
...      (Vectors.dense([0.5]), 0.0),
...      (Vectors.dense([0.6]), 1.0),
...      (Vectors.dense([1.0]), 1.0)] * 10,
...     ["features", "label"])
>>> lr = LogisticRegression()
>>> grid = ParamGridBuilder().addGrid(lr.maxIter, [0, 1]).build()
>>> evaluator = BinaryClassificationEvaluator()
>>> cv = CrossValidator(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator,
...     parallelism=2)
>>> cvModel = cv.fit(dataset)
>>> cvModel.getNumFolds()
3
>>> cvModel.avgMetrics[0]
0.5
>>> path = tempfile.mkdtemp()
>>> model_path = path + "/model"
>>> cvModel.write().save(model_path)
>>> cvModelRead = CrossValidatorModel.read().load(model_path)
>>> cvModelRead.avgMetrics
[0.5, ...
>>> evaluator.evaluate(cvModel.transform(dataset))
0.8333...
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

collectSubModels = *Param(parent='undefined', name='collectSubModels', doc='Param for whether to collect a list of sub-models trained during tuning. If set to false, then only the single best sub-model will be available after fitting. If set to true, then all sub-models will be available. Warning: For large models, collecting all sub-models can cause OOMs on the Spark driver.')*

copy(*extra=None*) [source]

Creates a copy of this instance with a randomly generated uid and some extra params. This copies creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

New in version 1.4.0.

estimator = *Param(parent='undefined', name='estimator', doc='estimator to be cross-validated')*

estimatorParamMaps = *Param(parent='undefined', name='estimatorParamMaps', doc='estimator param maps')*

evaluator = *Param(parent='undefined', name='evaluator', doc='evaluator used to select hyper-parameters that maximize the validator metric')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]*. *index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getCollectSubModels()

Gets the value of collectSubModels or its default value.

getEstimator()

Gets the value of estimator or its default value.

New in version 2.0.0.

getEstimatorParamMaps()

Gets the value of estimatorParamMaps or its default value.

New in version 2.0.0.

getEvaluator()
Gets the value of evaluator or its default value.

New in version 2.0.0.

getNumFolds()
Gets the value of numFolds or its default value.

New in version 1.4.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParallelism()
Gets the value of parallelism or its default value.

getParam(*paramName*)
Gets a param by its name.

getSeed()
Gets the value of seed or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numFolds = *Param(parent='undefined', name='numFolds', doc='number of folds for cross validation')*

parallelism = *Param(parent='undefined', name='parallelism', doc='the number of threads to use when running parallel algorithms (>= 1).')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**() [source]
Returns an MLReader instance for this class.

New in version 2.3.0.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)
Sets a parameter in the embedded param map.

setCollectSubModels(*value*) [source]
Sets the value of **collectSubModels**.

setEstimator(*value*) [source]
Sets the value of **estimator**.

New in version 2.0.0.

setEstimatorParamMaps(*value*) [source]
Sets the value of **estimatorParamMaps**.

New in version 2.0.0.

setEvaluator(*value*) [source]
Sets the value of **evaluator**.

New in version 2.0.0.

setNumFolds(*value*) [source]
Sets the value of **numFolds**.

New in version 1.4.0.

setParallelism(*value*) [source]
Sets the value of **parallelism**.

setParams(*estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *numFolds=3*, *seed=None*, *parallelism=1*, *collectSubModels=False*) [source]
setParams(self, estimator=None, estimatorParamMaps=None, evaluator=None, numFolds=3, seed=None, parallelism=1, collectSubModels=False): Sets params for cross validator.

New in version 1.4.0.

setSeed(*value*) [source]
Sets the value of **seed**.

write() [source]
Returns an MLWriter instance for this ML instance.

New in version 2.3.0.

class pyspark.ml.tuning.**CrossValidatorModel**(*bestModel*, *avgMetrics=[]*, *subModels=None*) [source]

CrossValidatorModel contains the model with the highest average cross-validation metric across folds and uses this model to transform input data. CrossValidatorModel also tracks the metrics for each param map evaluated.

New in version 1.4.0.

avgMetrics = *None*

Average cross-validation metrics for each paramMap in CrossValidator.estimatorParamMaps, in the corresponding order.

bestModel = *None*

best model from cross validation

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*) [source]

Creates a copy of this instance with a randomly generated uid and some extra params. This copies the underlying bestModel, creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over. It does not copy the extra Params into the subModels.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

New in version 1.4.0.

estimator = *Param(parent='undefined', name='estimator', doc='estimator to be cross-validated')*

estimatorParamMaps = *Param(parent='undefined', name='estimatorParamMaps', doc='estimator param maps')*

evaluator = *Param(parent='undefined', name='evaluator', doc='evaluator used to select hyper-parameters that maximize the validator metric')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getEstimator()

Gets the value of estimator or its default value.

New in version 2.0.0.

getEstimatorParamMaps()

Gets the value of estimatorParamMaps or its default value.

New in version 2.0.0.

getEvaluator()

Gets the value of evaluator or its default value.

New in version 2.0.0.

getNumFolds()

Gets the value of numFolds or its default value.

New in version 1.4.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getSeed()

Gets the value of seed or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

numFolds = *Param(parent='undefined', name='numFolds', doc='number of folds for cross validation')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**() [source]

Returns an MLReader instance for this class.

New in version 2.3.0.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setEstimator(*value*) [source]

Sets the value of **estimator**.

New in version 2.0.0.

setEstimatorParamMaps(*value*) [source]

Sets the value of **estimatorParamMaps**.

New in version 2.0.0.

setEvaluator(*value*) [source]

Sets the value of **evaluator**.

New in version 2.0.0.

subModels = *None*

sub model list from cross validation

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:
transformed dataset

New in version 1.3.0.

write() [source]

Returns an MLWriter instance for this ML instance.

New in version 2.3.0.

class pyspark.ml.tuning.**TrainValidationSplit**(*estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *trainRatio=0.75*, *parallelism=1*, *collectSubModels=False*, *seed=None*) [source]

Validation for hyper-parameter tuning. Randomly splits the input dataset into train and validation sets, and uses evaluation metric on the validation set to select the best model. Similar to **CrossValidator**, but only splits the set once.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.tuning import TrainValidationSplitModel
>>> import tempfile
>>> dataset = spark.createDataFrame(
...     [(Vectors.dense([0.0]), 0.0),
...      (Vectors.dense([0.4]), 1.0),
...      (Vectors.dense([0.5]), 0.0),
...      (Vectors.dense([0.6]), 1.0),
...      (Vectors.dense([1.0]), 1.0)] * 10,
...     ["features", "label"]).repartition(1)
>>> lr = LogisticRegression()
>>> grid = ParamGridBuilder().addGrid(lr.maxIter, [0, 1]).build()
>>> evaluator = BinaryClassificationEvaluator()
>>> tvs = TrainValidationSplit(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator,
...     parallelism=1, seed=42)
>>> tvsModel = tvs.fit(dataset)
>>> tvsModel.getTrainRatio()
0.75
>>> tvsModel.validationMetrics
[0.5, ...
>>> path = tempfile.mkdtemp()
>>> model_path = path + "/model"
>>> tvsModel.write().save(model_path)
>>> tvsModelRead = TrainValidationSplitModel.read().load(model_path)
>>> tvsModelRead.validationMetrics
[0.5, ...
>>> evaluator.evaluate(tvsModel.transform(dataset))
0.833...
```

New in version 2.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

collectSubModels = *Param(parent='undefined', name='collectSubModels', doc='Param for whether to collect a list of sub-models trained during tuning. If set to false, then only the single best sub-model will be available after fitting. If set to true, then all sub-models will be available. Warning: For large models, collecting all sub-models can cause OOMs on the Spark driver.')*

copy(*extra=None*) [source]

Creates a copy of this instance with a randomly generated uid and some extra params. This copies creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

New in version 2.0.0.

estimator = *Param(parent='undefined', name='estimator', doc='estimator to be cross-validated')*

estimatorParamMaps = *Param(parent='undefined', name='estimatorParamMaps', doc='estimator param maps')*

evaluator = *Param(parent='undefined', name='evaluator', doc='evaluator used to select hyper-parameters that maximize the validator metric')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getCollectSubModels()

Gets the value of collectSubModels or its default value.

getEstimator()

Gets the value of estimator or its default value.

New in version 2.0.0.

getEstimatorParamMaps()

Gets the value of estimatorParamMaps or its default value.

New in version 2.0.0.

getEvaluator()

Gets the value of evaluator or its default value.

New in version 2.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParallelism()

Gets the value of parallelism or its default value.

getParam(*paramName*)

Gets a param by its name.

getSeed()

Gets the value of seed or its default value.

getTrainRatio()

Gets the value of trainRatio or its default value.

New in version 2.0.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

parallelism = *Param(parent='undefined', name='parallelism', doc='the number of threads to use when running parallel algorithms (>= 1).')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**() [source]

Returns an MLReader instance for this class.

New in version 2.3.0.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setCollectSubModels(*value*) [source]

Sets the value of **collectSubModels**.

setEstimator(*value*) [source]

Sets the value of **estimator**.

New in version 2.0.0.

setEstimatorParamMaps(*value*) [source]

Sets the value of **estimatorParamMaps**.

New in version 2.0.0.

setEvaluator(*value*) [source]

Sets the value of **evaluator**.

New in version 2.0.0.

setParallelism(*value*) [source]

Sets the value of **parallelism**.

setParams(*estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *trainRatio=0.75*, *parallelism=1*, *collectSubModels=False*, *seed=None*) [source]

setParams(self, estimator=None, estimatorParamMaps=None, evaluator=None, trainRatio=0.75, parallelism=1, collectSubModels=False, seed=None): Sets params for the train validation split.

New in version 2.0.0.

setSeed(*value*) [source]

Sets the value of **seed**.

setTrainRatio(*value*) [source]

Sets the value of **trainRatio**.

New in version 2.0.0.

trainRatio = *Param(parent='undefined', name='trainRatio', doc='Param for ratio between train and validation data. Must be between 0 and 1.')*

write() [source]

Returns an MLWriter instance for this ML instance.

New in version 2.3.0.

class pyspark.ml.tuning.**TrainValidationSplitModel**(*bestModel*, *validationMetrics=[]*, *subModels=None*) [source]

Model from train validation split.

New in version 2.0.0.

bestModel = *None*

best model from train validation split

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*) [source]

Creates a copy of this instance with a randomly generated uid and some extra params. This copies the underlying bestModel, creates a deep copy of the embedded paramMap, and copies the embedded and extra parameters over. And, this creates a shallow copy of the validationMetrics. It does not copy the extra Params into the subModels.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

New in version 2.0.0.

estimator = *Param(parent='undefined', name='estimator', doc='estimator to be cross-validated')*

estimatorParamMaps = *Param(parent='undefined', name='estimatorParamMaps', doc='estimator param maps')*

evaluator = *Param(parent='undefined', name='evaluator', doc='evaluator used to select hyper-parameters that maximize the validator metric')*

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getEstimator()

Gets the value of estimator or its default value.

New in version 2.0.0.

getEstimatorParamMaps()

Gets the value of estimatorParamMaps or its default value.

New in version 2.0.0.

getEvaluator()

Gets the value of evaluator or its default value.

New in version 2.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getSeed()

Gets the value of seed or its default value.

getTrainRatio()

Gets the value of trainRatio or its default value.

New in version 2.0.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

classmethod **read**() [source]

Returns an MLReader instance for this class.

New in version 2.3.0.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

seed = *Param(parent='undefined', name='seed', doc='random seed.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setEstimator(*value*) [source]

Sets the value of **estimator**.

New in version 2.0.0.

setEstimatorParamMaps(*value*) [source]

Sets the value of **estimatorParamMaps**.

New in version 2.0.0.

setEvaluator(*value*) [source]

Sets the value of **evaluator**.

New in version 2.0.0.

subModels = *None*

sub models from train validation split

trainRatio = *Param(parent='undefined', name='trainRatio', doc='Param for ratio between train and validation data. Must be between 0 and 1.')*

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:

- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

validationMetrics = *None*

evaluated validation metrics

write() [source]

Returns an MLWriter instance for this ML instance.

New in version 2.3.0.

pyspark.ml.evaluation module

class pyspark.ml.evaluation.**Evaluator** [source]

Base class for evaluators that compute metrics from predictions.

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. The default implementation creates a shallow copy using **copy.copy()**, and then copies the embedded and extra parameters over and returns the copy. Subclasses should override this method if the default approach is not sufficient.

Parameters:

extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

evaluate(*dataset*, *params=None*) [source]

Evaluates the output with optional parameters.

Parameters:
- **dataset** – a dataset that contains labels/observations and predictions
- **params** – an optional param map that overrides embedded params

Returns:
metric

New in version 1.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isLargerBetter() [source]

Indicates whether the metric returned by **evaluate()** should be maximized (True, default) or minimized (False). A given evaluator may support multiple metrics which may be maximized or minimized.

New in version 1.5.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

set(*param*, *value*)

Sets a parameter in the embedded param map.

class pyspark.ml.evaluation.**BinaryClassificationEvaluator**(*rawPredictionCol='rawPrediction'*, *labelCol='label'*, *metricName='areaUnderROC'*, *weightCol=None*, *numBins=1000*) [source]

Evaluator for binary classification, which expects two input columns: rawPrediction and label. The rawPrediction column can be of type double (binary 0/1 prediction, or probability of label 1) or of type vector (length-2 vector of raw predictions, scores, or label probabilities).

```
>>> from pyspark.ml.linalg import Vectors
>>> scoreAndLabels = map(lambda x: (Vectors.dense([1.0 - x[0], x[0]]), x[1]),
...    [(0.1, 0.0), (0.1, 1.0), (0.4, 0.0), (0.6, 0.0), (0.6, 1.0), (0.6, 1.0), (0.8, 1.0)])
>>> dataset = spark.createDataFrame(scoreAndLabels, ["raw", "label"])
...
>>> evaluator = BinaryClassificationEvaluator()
>>> evaluator.setRawPredictionCol("raw")
BinaryClassificationEvaluator...
>>> evaluator.evaluate(dataset)
0.70...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "areaUnderPR"})
0.83...
>>> bce_path = temp_path + "/bce"
>>> evaluator.save(bce_path)
>>> evaluator2 = BinaryClassificationEvaluator.load(bce_path)
>>> str(evaluator2.getRawPredictionCol())
'raw'
>>> scoreAndLabelsAndWeight = map(lambda x: (Vectors.dense([1.0 - x[0], x[0]]), x[1], x[2]),
...    [(0.1, 0.0, 1.0), (0.1, 1.0, 0.9), (0.4, 0.0, 0.7), (0.6, 0.0, 0.9),
...     (0.6, 1.0, 1.0), (0.6, 1.0, 0.3), (0.8, 1.0, 1.0)])
>>> dataset = spark.createDataFrame(scoreAndLabelsAndWeight, ["raw", "label", "weight"])
...
>>> evaluator = BinaryClassificationEvaluator(rawPredictionCol="raw", weightCol="weight")
>>> evaluator.evaluate(dataset)
0.70...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "areaUnderPR"})
0.82...
>>> evaluator.getNumBins()
1000
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance

Returns:
Copy of this instance

evaluate(*dataset*, *params=None*)

Evaluates the output with optional parameters.

Parameters:
- **dataset** – a dataset that contains labels/observations and predictions
- **params** – an optional param map that overrides embedded params

Returns:
metric

New in version 1.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getLabelCol()

Gets the value of labelCol or its default value.

getMetricName() [source]

Gets the value of metricName or its default value.

New in version 1.4.0.

getNumBins() [source]

Gets the value of numBins or its default value.

New in version 3.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getRawPredictionCol()

Gets the value of rawPredictionCol or its default value.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isLargerBetter()

Indicates whether the metric returned by **evaluate()** should be maximized (True, default) or minimized (False). A given evaluator may support multiple metrics which may be maximized or minimized.

New in version 1.5.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

metricName = *Param(parent='undefined', name='metricName', doc='metric name in evaluation (areaUnderROC|areaUnderPR)')*

numBins = *Param(parent='undefined', name='numBins', doc='Number of bins to down-sample the curves (ROC curve, PR curve) in area computation. If 0, no down-sampling will occur. Must be >= 0.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

rawPredictionCol = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setLabelCol(*value*) [source]

Sets the value of **labelCol**.

setMetricName(*value*) [source]

Sets the value of **metricName**.

New in version 1.4.0.

setNumBins(*value*) [source]

Sets the value of **numBins**.

New in version 3.0.0.

setParams(*self*, *rawPredictionCol="rawPrediction"*, *labelCol="label"*, *metricName="areaUnderROC"*, *weightCol=None*, *numBins=1000*) [source]

Sets params for binary classification evaluator.

New in version 1.4.0.

setRawPredictionCol(*value*) [source]

Sets the value of **rawPredictionCol**.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

New in version 3.0.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.evaluation.**RegressionEvaluator**(*predictionCol='prediction'*, *labelCol='label'*, *metricName='rmse'*, *weightCol=None*, *throughOrigin=False*) [source]

Evaluator for Regression, which expects input columns prediction, label and an optional weight column.

```
>>> scoreAndLabels = [(-28.98343821, -27.0), (20.21491975, 21.5),
...   (-25.98418959, -22.0), (30.69731842, 33.0), (74.69283752, 71.0)]
>>> dataset = spark.createDataFrame(scoreAndLabels, ["raw", "label"])
...
>>> evaluator = RegressionEvaluator()
>>> evaluator.setPredictionCol("raw")
RegressionEvaluator...
>>> evaluator.evaluate(dataset)
2.842...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "r2"})
0.993...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "mae"})
2.649...
>>> re_path = temp_path + "/re"
>>> evaluator.save(re_path)
>>> evaluator2 = RegressionEvaluator.load(re_path)
>>> str(evaluator2.getPredictionCol())
'raw'
>>> scoreAndLabelsAndWeight = [(-28.98343821, -27.0, 1.0), (20.21491975, 21.5, 0.8),
...   (-25.98418959, -22.0, 1.0), (30.69731842, 33.0, 0.6), (74.69283752, 71.0, 0.2)]
>>> dataset = spark.createDataFrame(scoreAndLabelsAndWeight, ["raw", "label", "weight"])
...
>>> evaluator = RegressionEvaluator(predictionCol="raw", weightCol="weight")
>>> evaluator.evaluate(dataset)
2.740...
>>> evaluator.getThroughOrigin()
False
```

New in version 1.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

evaluate(*dataset*, *params=None*)

Evaluates the output with optional parameters.

Parameters:
- **dataset** – a dataset that contains labels/observations and predictions
- **params** – an optional param map that overrides embedded params

Returns:
metric

New in version 1.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getLabelCol()

Gets the value of labelCol or its default value.

getMetricName() [source]

Gets the value of metricName or its default value.

New in version 1.4.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

getThroughOrigin() [source]

Gets the value of throughOrigin or its default value.

New in version 3.0.0.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isLargerBetter()

Indicates whether the metric returned by **evaluate()** should be maximized (True, default) or minimized (False). A given evaluator may support multiple metrics which may be maximized or minimized.

New in version 1.5.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

metricName = *Param(parent='undefined', name='metricName', doc='metric name in evaluation - one of:\n rmse - root mean squared error (default)\n mse - mean squared error\n r2 - r^2 metric\n mae - mean absolute error\n var - explained variance.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setLabelCol(*value*) [source]

Sets the value of **labelCol**.

setMetricName(*value*) [source]

Sets the value of **metricName**.

New in version 1.4.0.

setParams(*self*, *predictionCol="prediction"*, *labelCol="label"*, *metricName="rmse"*, *weightCol=None*, *throughOrigin=False*) [source]

Sets params for regression evaluator.

New in version 1.4.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

setThroughOrigin(*value*) [source]

Sets the value of **throughOrigin**.

New in version 3.0.0.

setWeightCol(*value*) [source]

Sets the value of **weightCol**.

New in version 3.0.0.

throughOrigin = *Param(parent='undefined', name='throughOrigin', doc='whether the regression is through the origin.')*

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.evaluation.**MulticlassClassificationEvaluator**(*predictionCol='prediction'*, *labelCol='label'*, *metricName='f1'*, *weightCol=None*, *metricLabel=0.0*, *beta=1.0*, *probabilityCol='probability'*, *eps=1e-15*) [source]

Evaluator for Multiclass Classification, which expects input columns: prediction, label, weight (optional) and probabilityCol (only for logLoss).

```
>>> scoreAndLabels = [(0.0, 0.0), (0.0, 1.0), (0.0, 0.0),
...     (1.0, 0.0), (1.0, 1.0), (1.0, 1.0), (1.0, 1.0), (2.0, 2.0), (2.0, 0.0)]
>>> dataset = spark.createDataFrame(scoreAndLabels, ["prediction", "label"])
>>> evaluator = MulticlassClassificationEvaluator()
>>> evaluator.setPredictionCol("prediction")
MulticlassClassificationEvaluator...
>>> evaluator.evaluate(dataset)
0.66...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "accuracy"})
0.66...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "truePositiveRateByLabel",
...     evaluator.metricLabel: 1.0})
0.75...
>>> evaluator.setMetricName("hammingLoss")
MulticlassClassificationEvaluator...
>>> evaluator.evaluate(dataset)
0.33...
>>> mce_path = temp_path + "/mce"
>>> evaluator.save(mce_path)
>>> evaluator2 = MulticlassClassificationEvaluator.load(mce_path)
>>> str(evaluator2.getPredictionCol())
'prediction'
>>> scoreAndLabelsAndWeight = [(0.0, 0.0, 1.0), (0.0, 1.0, 1.0), (0.0, 0.0, 1.0),
...     (1.0, 0.0, 1.0), (1.0, 1.0, 1.0), (1.0, 1.0, 1.0), (1.0, 1.0, 1.0),
...     (2.0, 2.0, 1.0), (2.0, 0.0, 1.0)]
>>> dataset = spark.createDataFrame(scoreAndLabelsAndWeight, ["prediction", "label", "weight"])
>>> evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",
...     weightCol="weight")
>>> evaluator.evaluate(dataset)
0.66...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "accuracy"})
0.66...
>>> predictionAndLabelsWithProbabilities = [
...     (1.0, 1.0, 1.0, [0.1, 0.8, 0.1]), (0.0, 2.0, 1.0, [0.9, 0.05, 0.05]),
...     (0.0, 0.0, 1.0, [0.8, 0.2, 0.0]), (1.0, 1.0, 1.0, [0.3, 0.65, 0.05])]
>>> dataset = spark.createDataFrame(predictionAndLabelsWithProbabilities, ["prediction",
...     "label", "weight", "probability"])
>>> evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",
...     probabilityCol="probability")
>>> evaluator.setMetricName("logLoss")
MulticlassClassificationEvaluator...
>>> evaluator.evaluate(dataset)
0.9682...
```

New in version 1.5.0.

beta = *Param(parent='undefined', name='beta', doc='The beta value used in weightedFMeasure|fMeasureByLabel. Must be > 0. The default value is 1.')*

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

eps = *Param(parent='undefined', name='eps', doc='log-loss is undefined for p=0 or p=1, so probabilities are clipped to max(eps, min(1 - eps, p)). Must be in range (0, 0.5). The default value is 1e-15.')*

evaluate(*dataset*, *params=None*)

Evaluates the output with optional parameters.

Parameters:
- **dataset** – a dataset that contains labels/observations and predictions
- **params** – an optional param map that overrides embedded params

Returns:
metric

New in version 1.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getBeta() [source]

Gets the value of beta or its default value.

New in version 3.0.0.

getEps() [source]

Gets the value of eps or its default value.

New in version 3.0.0.

getLabelCol()

Gets the value of labelCol or its default value.

getMetricLabel() [source]

Gets the value of metricLabel or its default value.

New in version 3.0.0.

getMetricName() [source]

Gets the value of metricName or its default value.

New in version 1.5.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

getProbabilityCol()
Gets the value of probabilityCol or its default value.

getWeightCol()
Gets the value of weightCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isLargerBetter()
Indicates whether the metric returned by **evaluate()** should be maximized (True, default) or minimized (False). A given evaluator may support multiple metrics which may be maximized or minimized.

New in version 1.5.0.

isSet(*param*)
Checks whether a param is explicitly set by user.

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

metricLabel = *Param(parent='undefined', name='metricLabel', doc='The class whose metric will be computed in truePositiveRateByLabel|falsePositiveRateByLabel|precisionByLabel|recallByLabel|fMeasureByLabel. Must be >= 0. The default value is 0.')*

metricName = *Param(parent='undefined', name='metricName', doc='metric name in evaluation (f1|accuracy|weightedPrecision|weightedRecall|weightedTruePositiveRate|weightedFalsePositiveRate|weightedFMeasure|truePositiveRateByLabel|falsePositiveRateByLabel|prec*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

probabilityCol = *Param(parent='undefined', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities should be treated as confidences, not precise probabilities.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setBeta(*value*) [source]
Sets the value of **beta**.

New in version 3.0.0.

setEps(*value*) [source]
Sets the value of **eps**.

New in version 3.0.0.

setLabelCol(*value*) [source]
Sets the value of **labelCol**.

setMetricLabel(*value*) [source]
Sets the value of **metricLabel**.

New in version 3.0.0.

setMetricName(*value*) [source]
Sets the value of **metricName**.

New in version 1.5.0.

setParams(*self*, *predictionCol="prediction"*, *labelCol="label"*, *metricName="f1"*, *weightCol=None*, *metricLabel=0.0*, *beta=1.0*, *probabilityCol="probability"*, *eps=1e-15*) [source]
Sets params for multiclass classification evaluator.

New in version 1.5.0.

setPredictionCol(*value*) [source]
Sets the value of **predictionCol**.

setProbabilityCol(*value*) [source]
Sets the value of **probabilityCol**.

New in version 3.0.0.

setWeightCol(*value*) [source]
Sets the value of **weightCol**.

New in version 3.0.0.

weightCol = *Param(parent='undefined', name='weightCol', doc='weight column name. If this is not set or empty, we treat all instance weights as 1.0.')*

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.evaluation.**MultilabelClassificationEvaluator**(*predictionCol='prediction'*, *labelCol='label'*, *metricName='f1Measure'*, *metricLabel=0.0*) [source]

Note: Experimental

Evaluator for Multilabel Classification, which expects two input columns: prediction and label.

```
>>> scoreAndLabels = [([0.0, 1.0], [0.0, 2.0]), ([0.0, 2.0], [0.0, 1.0]),
...     ([], [0.0]), ([2.0], [2.0]), ([2.0, 0.0], [2.0, 0.0]),
...     ([0.0, 1.0, 2.0], [0.0, 1.0]), ([1.0], [1.0, 2.0])]
>>> dataset = spark.createDataFrame(scoreAndLabels, ["prediction", "label"])
...
>>> evaluator = MultilabelClassificationEvaluator()
>>> evaluator.setPredictionCol("prediction")
MultilabelClassificationEvaluator...
>>> evaluator.evaluate(dataset)
0.63...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "accuracy"})
0.54...
>>> mlce_path = temp_path + "/mlce"
>>> evaluator.save(mlce_path)
>>> evaluator2 = MultilabelClassificationEvaluator.load(mlce_path)
>>> str(evaluator2.getPredictionCol())
'prediction'
```

New in version 3.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

evaluate(*dataset*, *params=None*)

Evaluates the output with optional parameters.

Parameters:
- **dataset** – a dataset that contains labels/observations and predictions
- **params** – an optional param map that overrides embedded params

Returns:
metric

New in version 1.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getLabelCol()

Gets the value of labelCol or its default value.

getMetricLabel() [source]

Gets the value of metricLabel or its default value.

New in version 3.0.0.

getMetricName() [source]

Gets the value of metricName or its default value.

New in version 3.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isLargerBetter()

Indicates whether the metric returned by **`evaluate()`** should be maximized (True, default) or minimized (False). A given evaluator may support multiple metrics which may be maximized or minimized.

New in version 1.5.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

`labelCol` = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **`load`**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

`metricLabel` = *Param(parent='undefined', name='metricLabel', doc='The class whose metric will be computed in precisionByLabel|recallByLabel|f1MeasureByLabel. Must be >= 0. The default value is 0.')*

`metricName` = *Param(parent='undefined', name='metricName', doc='metric name in evaluation (subsetAccuracy|accuracy|hammingLoss|precision|recall|f1Measure|precisionByLabel|recallByLabel|f1MeasureByLabel|microPrecision|microRecall|microF1Measure)')*

property **`params`**

Returns all params ordered by name. The default implementation uses **`dir()`** to get all attributes of type **`Param`**.

`predictionCol` = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **`read`**()

Returns an MLReader instance for this class.

`save`(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

`set`(*param*, *value*)

Sets a parameter in the embedded param map.

`setLabelCol`(*value*) [source]

Sets the value of **`labelCol`**.

New in version 3.0.0.

`setMetricLabel`(*value*) [source]

Sets the value of **`metricLabel`**.

New in version 3.0.0.

`setMetricName`(*value*) [source]

Sets the value of **`metricName`**.

New in version 3.0.0.

`setParams`(*self*, *predictionCol="prediction"*, *labelCol="label"*, *metricName="f1Measure"*, *metricLabel=0.0*) [source]

Sets params for multilabel classification evaluator.

New in version 3.0.0.

`setPredictionCol`(*value*) [source]

Sets the value of **`predictionCol`**.

New in version 3.0.0.

`write`()

Returns an MLWriter instance for this ML instance.

class `pyspark.ml.evaluation.`**`ClusteringEvaluator`**(*predictionCol='prediction'*, *featuresCol='features'*, *metricName='silhouette'*, *distanceMeasure='squaredEuclidean'*) [source]

Evaluator for Clustering results, which expects two input columns: prediction and features. The metric computes the Silhouette measure using the squared Euclidean distance.

The Silhouette is a measure for the validation of the consistency within clusters. It ranges between 1 and -1, where a value close to 1 means that the points in a cluster are close to the other points in the same cluster and far from the points of the other clusters.

```
>>> from pyspark.ml.linalg import Vectors
>>> featureAndPredictions = map(lambda x: (Vectors.dense(x[0]), x[1]),
...     [([0.0, 0.5], 0.0), ([0.5, 0.0], 0.0), ([10.0, 11.0], 1.0),
...     ([10.5, 11.5], 1.0), ([1.0, 1.0], 0.0), ([8.0, 6.0], 1.0)])
>>> dataset = spark.createDataFrame(featureAndPredictions, ["features", "prediction"])
...
>>> evaluator = ClusteringEvaluator()
>>> evaluator.setPredictionCol("prediction")
ClusteringEvaluator...
>>> evaluator.evaluate(dataset)
0.9079...
>>> ce_path = temp_path + "/ce"
>>> evaluator.save(ce_path)
>>> evaluator2 = ClusteringEvaluator.load(ce_path)
>>> str(evaluator2.getPredictionCol())
'prediction'
```

New in version 2.3.0.

`clear`(*param*)

Clears a param from the param map if it has been explicitly set.

`copy`(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

`distanceMeasure` = *Param(parent='undefined', name='distanceMeasure', doc="The distance measure. Supported options: 'squaredEuclidean' and 'cosine'.")*

`evaluate`(*dataset*, *params=None*)

Evaluates the output with optional parameters.

Parameters:
- **dataset** – a dataset that contains labels/observations and predictions
- **params** – an optional param map that overrides embedded params

Returns:
metric

New in version 1.4.0.

explainParam(*param*)
Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()
Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)
Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values

Returns:
merged param map

featuresCol = *Param(parent='undefined', name='featuresCol', doc='features column name.')*

getDistanceMeasure() [source]
Gets the value of *distanceMeasure*

New in version 2.4.0.

getFeaturesCol()
Gets the value of featuresCol or its default value.

getMetricName() [source]
Gets the value of metricName or its default value.

New in version 2.3.0.

getOrDefault(*param*)
Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)
Gets a param by its name.

getPredictionCol()
Gets the value of predictionCol or its default value.

hasDefault(*param*)
Checks whether a param has a default value.

hasParam(*paramName*)
Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)
Checks whether a param is explicitly set by user or has a default value.

isLargerBetter()
Indicates whether the metric returned by **evaluate()** should be maximized (True, default) or minimized (False). A given evaluator may support multiple metrics which may be maximized or minimized.

New in version 1.5.0.

isSet(*param*)
Checks whether a param is explicitly set by user.

classmethod **load**(*path*)
Reads an ML instance from the input path, a shortcut of *read().load(path)*.

metricName = *Param(parent='undefined', name='metricName', doc='metric name in evaluation (silhouette)')*

property **params**
Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()
Returns an MLReader instance for this class.

save(*path*)
Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)
Sets a parameter in the embedded param map.

setDistanceMeasure(*value*) [source]
Sets the value of **distanceMeasure**.

New in version 2.4.0.

setFeaturesCol(*value*) [source]
Sets the value of **featuresCol**.

setMetricName(*value*) [source]
Sets the value of **metricName**.

New in version 2.3.0.

setParams(*self*, *predictionCol="prediction"*, *featuresCol="features"*, *metricName="silhouette"*, *distanceMeasure="squaredEuclidean"*) [source]
Sets params for clustering evaluator.

New in version 2.3.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.evaluation.**RankingEvaluator**(*predictionCol='prediction'*, *labelCol='label'*, *metricName='meanAveragePrecision'*, *k=10*) [source]

Note: Experimental

Evaluator for Ranking, which expects two input columns: prediction and label.

```
>>> scoreAndLabels = [([1.0, 6.0, 2.0, 7.0, 8.0, 3.0, 9.0, 10.0, 4.0, 5.0],
...     [1.0, 2.0, 3.0, 4.0, 5.0]),
...     ([4.0, 1.0, 5.0, 6.0, 2.0, 7.0, 3.0, 8.0, 9.0, 10.0], [1.0, 2.0, 3.0]),
...     ([1.0, 2.0, 3.0, 4.0, 5.0], [])]
>>> dataset = spark.createDataFrame(scoreAndLabels, ["prediction", "label"])
...
>>> evaluator = RankingEvaluator()
>>> evaluator.setPredictionCol("prediction")
RankingEvaluator...
>>> evaluator.evaluate(dataset)
0.35...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "precisionAtK", evaluator.k: 2})
0.33...
>>> ranke_path = temp_path + "/ranke"
>>> evaluator.save(ranke_path)
>>> evaluator2 = RankingEvaluator.load(ranke_path)
>>> str(evaluator2.getPredictionCol())
'prediction'
```

New in version 3.0.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

evaluate(*dataset*, *params=None*)

Evaluates the output with optional parameters.

Parameters:
- **dataset** – a dataset that contains labels/observations and predictions
- **params** – an optional param map that overrides embedded params

Returns:
metric

New in version 1.4.0.

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

getK() [source]

Gets the value of k or its default value.

New in version 3.0.0.

getLabelCol()

Gets the value of labelCol or its default value.

getMetricName() [source]

Gets the value of metricName or its default value.

New in version 3.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isLargerBetter()

Indicates whether the metric returned by **evaluate()** should be maximized (True, default) or minimized (False). A

given evaluator may support multiple metrics which may be maximized or minimized.

New in version 1.5.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

k = *Param(parent='undefined', name='k', doc='The ranking position value used in meanAveragePrecisionAtK|precisionAtK|ndcgAtK|recallAtK. Must be > 0. The default value is 10.')*

labelCol = *Param(parent='undefined', name='labelCol', doc='label column name.')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

metricName = *Param(parent='undefined', name='metricName', doc='metric name in evaluation (meanAveragePrecision|meanAveragePrecisionAtK|precisionAtK|ndcgAtK|recallAtK)')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setK(*value*) [source]

Sets the value of **k**.

New in version 3.0.0.

setLabelCol(*value*) [source]

Sets the value of **labelCol**.

New in version 3.0.0.

setMetricName(*value*) [source]

Sets the value of **metricName**.

New in version 3.0.0.

setParams(*self*, *predictionCol="prediction"*, *labelCol="label"*, *metricName="meanAveragePrecision"*, *k=10*) [source]

Sets params for ranking evaluator.

New in version 3.0.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 3.0.0.

write()

Returns an MLWriter instance for this ML instance.

pyspark.ml.fpm module

class pyspark.ml.fpm.**FPGrowth**(*minSupport=0.3*, *minConfidence=0.8*, *itemsCol='items'*, *predictionCol='prediction'*, *numPartitions=None*) [source]

A parallel FP-growth algorithm to mine frequent itemsets. The algorithm is described in Li et al., PFP: Parallel FP-Growth for Query Recommendation [LI2008]. PFP distributes computation in such a way that each worker executes an independent group of mining tasks. The FP-Growth algorithm is described in Han et al., Mining frequent patterns without candidate generation [HAN2000]

[LI2008]://doi.org/10.1145/1454008.1454027

[HAN2000]://doi.org/10.1145/335191.335372

Note: null values in the feature column are ignored during fit().

Note: Internally *transform collects* and *broadcasts* association rules.

```
>>> from pyspark.sql.functions import split
>>> data = (spark.read
...     .text("data/mllib/sample_fpgrowth.txt")
...     .select(split("value", "\s+").alias("items")))
>>> data.show(truncate=False)
+------------------------+
|items                   |
+------------------------+
|[r, z, h, k, p]         |
|[z, y, x, w, v, u, t, s]|
|[s, x, o, n, r]         |
|[x, z, y, m, t, s, q, e]|
|[z]                     |
|[x, z, y, r, q, t, p]   |
+------------------------+
...
>>> fp = FPGrowth(minSupport=0.2, minConfidence=0.7)
>>> fpm = fp.fit(data)
>>> fpm.setPredictionCol("newPrediction")
FPGrowthModel...
>>> fpm.freqItemsets.show(5)
+---------+----+
|    items|freq|
+---------+----+
|      [s]|   3|
|   [s, x]|   3|
|[s, x, z]|   2|
|   [s, z]|   2|
|      [r]|   3|
+---------+----+
only showing top 5 rows
...
>>> fpm.associationRules.show(5)
+----------+----------+----------+----+
|antecedent|consequent|confidence|lift|
+----------+----------+----------+----+
|    [t, s]|       [y]|       1.0| 2.0|
|    [t, s]|       [x]|       1.0| 1.5|
|    [t, s]|       [z]|       1.0| 1.2|
|       [p]|       [r]|       1.0| 2.0|
|       [p]|       [z]|       1.0| 1.2|
+----------+----------+----------+----+
only showing top 5 rows
...
>>> new_data = spark.createDataFrame([(["t", "s"], )], ["items"])
>>> sorted(fpm.transform(new_data).first().newPrediction)
['x', 'y', 'z']
```

New in version 2.2.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

fit(*dataset*, *params=None*)

Fits a model to the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params. If a list/tuple of param maps is given, this calls fit on each param map and returns a list of models.

Returns:
fitted model(s)

New in version 1.3.0.

fitMultiple(*dataset*, *paramMaps*)

Fits a model to the input dataset for each param map in *paramMaps*.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**.
- **paramMaps** – A Sequence of param maps.

Returns:
A thread safe iterable which contains one model for each param map. Each call to *next(modelIterator)* will return *(index, model)* where model was fit using *paramMaps[index]. index* values may not be sequential.

> **Note:** DeveloperApi

New in version 2.3.0.

getItemsCol()

Gets the value of itemsCol or its default value.

getMinConfidence()

Gets the value of minConfidence or its default value.

getMinSupport()

Gets the value of minSupport or its default value.

getNumPartitions()

Gets the value of **numPartitions** or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

itemsCol = *Param(parent='undefined', name='itemsCol', doc='items column name')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

minConfidence = *Param(parent='undefined', name='minConfidence', doc='Minimal confidence for generating Association Rule. [0.0, 1.0]. minConfidence will not affect the mining for frequent itemsets, but will affect the association rules generation.')*

minSupport = *Param(parent='undefined', name='minSupport', doc='Minimal support level of the frequent pattern. [0.0, 1.0]. Any pattern that appears more than (minSupport * size-of-the-dataset) times will be output in the frequent itemsets.')*

numPartitions = *Param(parent='undefined', name='numPartitions', doc='Number of partitions (at least 1) used by parallel FP-growth. By default the param is not set, and partition number of the input dataset is used.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setItemsCol(*value*) [source]

Sets the value of **itemsCol**.

setMinConfidence(*value*) [source]

Sets the value of **minConfidence**.

setMinSupport(*value*) [source]

Sets the value of **minSupport**.

setNumPartitions(*value*) [source]

Sets the value of **numPartitions**.

setParams(*self*, *minSupport=0.3*, *minConfidence=0.8*, *itemsCol="items"*, *predictionCol="prediction"*, *numPartitions=None*) [source]

New in version 2.2.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.fpm.**FPGrowthModel**(*java_model=None*) [source]

Model fitted by FPGrowth.

New in version 2.2.0.

property **associationRules**

DataFrame with four columns: * *antecedent* - Array of the same type as the input column. * *consequent* - Array of the same type as the input column. * *confidence* - Confidence for the rule (*DoubleType*). * *lift* - Lift for the rule (*DoubleType*).

New in version 2.2.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

property **freqItemsets**

DataFrame with two columns: * *items* - Itemset of the same type as the input column. * *freq* - Frequency of the itemset (*LongType*).

New in version 2.2.0.

getItemsCol()

Gets the value of itemsCol or its default value.

getMinConfidence()

Gets the value of minConfidence or its default value.

getMinSupport()

Gets the value of minSupport or its default value.

getNumPartitions()

Gets the value of **numPartitions** or its default value.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getPredictionCol()

Gets the value of predictionCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

itemsCol = *Param(parent='undefined', name='itemsCol', doc='items column name')*

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

minConfidence = *Param(parent='undefined', name='minConfidence', doc='Minimal confidence for generating Association Rule. [0.0, 1.0]. minConfidence will not affect the mining for frequent itemsets, but will affect the association rules generation.')*

minSupport = *Param(parent='undefined', name='minSupport', doc='Minimal support level of the frequent pattern. [0.0, 1.0]. Any pattern that appears more than (minSupport * size-of-the-dataset) times will be output in the frequent itemsets.')*

numPartitions = *Param(parent='undefined', name='numPartitions', doc='Number of partitions (at least 1) used by parallel FP-growth. By default the param is not set, and partition number of the input dataset is used.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

predictionCol = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **read**()

Returns an MLReader instance for this class.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

set(*param*, *value*)

Sets a parameter in the embedded param map.

setItemsCol(*value*) [source]

Sets the value of **itemsCol**.

New in version 3.0.0.

setMinConfidence(*value*) [source]

Sets the value of **minConfidence**.

New in version 3.0.0.

setPredictionCol(*value*) [source]

Sets the value of **predictionCol**.

New in version 3.0.0.

transform(*dataset*, *params=None*)

Transforms the input dataset with optional parameters.

Parameters:
- **dataset** – input dataset, which is an instance of **pyspark.sql.DataFrame**
- **params** – an optional param map that overrides embedded params.

Returns:

transformed dataset

New in version 1.3.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.fpm.**PrefixSpan**(*minSupport=0.1*, *maxPatternLength=10*, *maxLocalProjDBSize=32000000*, *sequenceCol='sequence'*) [source]

A parallel PrefixSpan algorithm to mine frequent sequential patterns. The PrefixSpan algorithm is described in J. Pei, et al., PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth (see here). This class is not yet an Estimator/Transformer, use **findFrequentSequentialPatterns()** method to run the PrefixSpan algorithm.

@see Sequential Pattern Mining (Wikipedia)

```
>>> from pyspark.ml.fpm import PrefixSpan
>>> from pyspark.sql import Row
>>> df = sc.parallelize([Row(sequence=[[1, 2], [3]]),
...                      Row(sequence=[[1], [3, 2], [1, 2]]),
...                      Row(sequence=[[1, 2], [5]]),
...                      Row(sequence=[[6]])]).toDF()
>>> prefixSpan = PrefixSpan()
>>> prefixSpan.getMaxLocalProjDBSize()
32000000
>>> prefixSpan.getSequenceCol()
'sequence'
>>> prefixSpan.setMinSupport(0.5)
PrefixSpan...
>>> prefixSpan.setMaxPatternLength(5)
PrefixSpan...
>>> prefixSpan.findFrequentSequentialPatterns(df).sort("sequence").show(truncate=False)
+----------+----+
|sequence  |freq|
+----------+----+
|[[1]]     |3   |
|[[1], [3]]|2   |
|[[2]]     |3   |
|[[2, 1]]  |3   |
|[[3]]     |2   |
+----------+----+
...
```

New in version 2.4.0.

clear(*param*)

Clears a param from the param map if it has been explicitly set.

copy(*extra=None*)

Creates a copy of this instance with the same uid and some extra params. This implementation first calls Params.copy and then make a copy of the companion Java pipeline component with extra params. So both the Python wrapper and the Java pipeline component get copied.

Parameters:
extra – Extra parameters to copy to the new instance
Returns:
Copy of this instance

explainParam(*param*)

Explains a single param and returns its name, doc, and optional default value and user-supplied value in a string.

explainParams()

Returns the documentation of all params with their optionally default values and user-supplied values.

extractParamMap(*extra=None*)

Extracts the embedded default param values and user-supplied values, and then merges them with extra values from input into a flat param map, where the latter value is used if there exist conflicts, i.e., with ordering: default param values < user-supplied values < extra.

Parameters:
extra – extra param values
Returns:
merged param map

findFrequentSequentialPatterns(*dataset*) [source]

Finds the complete set of frequent sequential patterns in the input sequences of itemsets.

Parameters:
dataset – A dataframe containing a sequence column which is *ArrayType(ArrayType(T))* type, T is the item type for the input dataset.
Returns:
A *DataFrame* that contains columns of sequence and corresponding frequency. The schema of it will be: - *sequence: ArrayType(ArrayType(T))* (T is the item type) - *freq: Long*

New in version 2.4.0.

New in version 2.4.0.

getMaxLocalProjDBSize() [source]

Gets the value of maxLocalProjDBSize or its default value.

New in version 3.0.0.

getMaxPatternLength() [source]

Gets the value of maxPatternLength or its default value.

New in version 3.0.0.

getMinSupport() [source]

Gets the value of minSupport or its default value.

New in version 3.0.0.

getOrDefault(*param*)

Gets the value of a param in the user-supplied param map or its default value. Raises an error if neither is set.

getParam(*paramName*)

Gets a param by its name.

getSequenceCol() [source]

Gets the value of sequenceCol or its default value.

New in version 3.0.0.

hasDefault(*param*)

Checks whether a param has a default value.

hasParam(*paramName*)

Tests whether this instance contains a param with a given (string) name.

isDefined(*param*)

Checks whether a param is explicitly set by user or has a default value.

isSet(*param*)

Checks whether a param is explicitly set by user.

maxLocalProjDBSize = *Param(parent='undefined', name='maxLocalProjDBSize', doc='The maximum number of items (including delimiters used in the internal storage format) allowed in a projected database before local processing. If a projected database exceeds this size, another iteration of distributed prefix growth is run. Must be > 0.')*

maxPatternLength = *Param(parent='undefined', name='maxPatternLength', doc='The maximal length of the sequential pattern. Must be > 0.')*

minSupport = *Param(parent='undefined', name='minSupport', doc='The minimal support level of the sequential pattern. Sequential pattern that appears more than (minSupport * size-of-the-dataset) times will be output. Must be >= 0.')*

property **params**

Returns all params ordered by name. The default implementation uses **dir()** to get all attributes of type **Param**.

sequenceCol = *Param(parent='undefined', name='sequenceCol', doc='The name of the sequence column in dataset, rows with nulls in this column are ignored.')*

set(*param*, *value*)

Sets a parameter in the embedded param map.

setMaxLocalProjDBSize(*value*) [source]

Sets the value of **maxLocalProjDBSize**.

New in version 3.0.0.

setMaxPatternLength(*value*) [source]

Sets the value of **maxPatternLength**.

New in version 3.0.0.

setMinSupport(*value*) [source]

Sets the value of **minSupport**.

New in version 3.0.0.

setParams(*self*, *minSupport=0.1*, *maxPatternLength=10*, *maxLocalProjDBSize=32000000*, *sequenceCol="sequence"*) [source]

New in version 2.4.0.

setSequenceCol(*value*) [source]

Sets the value of **sequenceCol**.

New in version 3.0.0.

pyspark.ml.image module

`pyspark.ml.image.`**ImageSchema**

An attribute of this module that contains the instance of **_ImageSchema**.

class `pyspark.ml.image.`**_ImageSchema** [source]

Internal class for *pyspark.ml.image.ImageSchema* attribute. Meant to be private and not to be instantized. Use *pyspark.ml.image.ImageSchema* attribute to access the APIs of this class.

property **columnSchema**

Returns the schema for the image column.

Returns:
a **StructType** for image column, `struct<origin:string, height:int, width:int, nChannels:int, mode:int, data:binary>`.

New in version 2.4.0.

property **imageFields**

Returns field names of image columns.

Returns:
a list of field names.

New in version 2.3.0.

property **imageSchema**

Returns the image schema.

Returns:
a **StructType** with a single column of images named "image" (nullable) and having the same type returned by **columnSchema()**.

New in version 2.3.0.

property **ocvTypes**

Returns the OpenCV type mapping supported.

Returns:
a dictionary containing the OpenCV type mapping supported.

New in version 2.3.0.

toImage(*array*, *origin=''*) [source]

Converts an array with metadata to a two-dimensional image.

Parameters:
- **array** (*numpy.ndarray*) – The array to convert to image.
- **origin** (*str*) – Path to the image, optional.

Returns:
a **`Row`** that is a two dimensional image.

New in version 2.3.0.

`toNDArray`(*image*) [source]

Converts an image to an array with metadata.

Parameters:
image (*Row*) – A row that contains the image to be converted. It should have the attributes specified in *ImageSchema.imageSchema*.

Returns:
a *numpy.ndarray* that is an image.

New in version 2.3.0.

property **`undefinedImageType`**

Returns the name of undefined image type for the invalid image.

New in version 2.3.0.

pyspark.ml.util module

class `pyspark.ml.util.`**`BaseReadWrite`** [source]

Base class for MLWriter and MLReader. Stores information about the SparkContext and SparkSession.

New in version 2.3.0.

property **`sc`**

Returns the underlying *SparkContext*.

`session`(*sparkSession*) [source]

Sets the Spark Session to use for saving/loading.

property **`sparkSession`**

Returns the user-specified Spark Session or the default.

class `pyspark.ml.util.`**`DefaultParamsReadable`** [source]

Note: DeveloperApi

Helper trait for making simple **`Params`** types readable. If a **`Params`** class stores all data as **`Param`** values, then extending this trait will provide a default implementation of reading saved instances of the class. This only handles simple **`Param`** types; e.g., it will not handle **`Dataset`**. See **`DefaultParamsWritable`**, the counterpart to this trait.

New in version 2.3.0.

classmethod **`load`**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

classmethod **`read`**() [source]

Returns a DefaultParamsReader instance for this class.

class `pyspark.ml.util.`**`DefaultParamsReader`**(*cls*) [source]

Note: DeveloperApi

Specialization of **`MLReader`** for **`Params`** types

Default **`MLReader`** implementation for transformers and estimators that contain basic (json-serializable) params and no data. This will not handle more complex params or types with data (e.g., models with coefficients).

New in version 2.3.0.

static **`getAndSetParams`**(*instance*, *metadata*) [source]

Extract Params from metadata, and set them in the instance.

`load`(*path*) [source]

Load the ML instance from the input path.

static **`loadMetadata`**(*path*, *sc*, *expectedClassName=''*) [source]

Load metadata saved using **`DefaultParamsWriter.saveMetadata()`**

Parameters:
expectedClassName – If non empty, this is checked against the loaded metadata.

static **`loadParamsInstance`**(*path*, *sc*) [source]

Load a **`Params`** instance from the given path, and return it. This assumes the instance inherits from **`MLReadable`**.

property **`sc`**

Returns the underlying *SparkContext*.

`session`(*sparkSession*)

Sets the Spark Session to use for saving/loading.

property **`sparkSession`**

Returns the user-specified Spark Session or the default.

class `pyspark.ml.util.`**`DefaultParamsWritable`** [source]

Note: DeveloperApi

Helper trait for making simple **`Params`** types writable. If a **`Params`** class stores all data as **`Param`** values, then extending this trait will provide a default implementation of writing saved instances of the class. This only handles simple **`Param`** types; e.g., it will not handle **`Dataset`**. See **`DefaultParamsReadable`**, the counterpart to this trait.

New in version 2.3.0.

`save`(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

write() [source]

Returns a DefaultParamsWriter instance for this class.

class pyspark.ml.util.**DefaultParamsWriter**(*instance*) [source]

Note: DeveloperApi

Specialization of **MLWriter** for **Params** types

Class for writing Estimators and Transformers whose parameters are JSON-serializable.

New in version 2.3.0.

overwrite()

Overwrites if the output path already exists.

save(*path*)

Save the ML instance to the input path.

saveImpl(*path*) [source]

save() handles overwriting and then calls this method. Subclasses should override this method to implement the actual saving of the instance.

static **saveMetadata**(*instance*, *path*, *sc*, *extraMetadata=None*, *paramMap=None*) [source]

Saves metadata + Params to: path + "/metadata" - class - timestamp - sparkVersion - uid - paramMap - defaultParamMap (since 2.4.0) - (optionally, extra metadata) :param extraMetadata: Extra metadata to be saved at same level as uid, paramMap, etc. :param paramMap: If given, this is saved in the "paramMap" field.

property **sc**

Returns the underlying *SparkContext*.

session(*sparkSession*)

Sets the Spark Session to use for saving/loading.

property **sparkSession**

Returns the user-specified Spark Session or the default.

class pyspark.ml.util.**GeneralJavaMLWritable** [source]

(Private) Mixin for ML instances that provide **GeneralJavaMLWriter**.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

write() [source]

Returns an GeneralMLWriter instance for this ML instance.

class pyspark.ml.util.**GeneralJavaMLWriter**(*instance*) [source]

(Private) Specialization of **GeneralMLWriter** for **JavaParams** types

format(*source*) [source]

Specifies the format of ML export (e.g. "pmml", "internal", or the fully qualified class name for export).

option(*key*, *value*)

overwrite()

Overwrites if the output path already exists.

save(*path*)

Save the ML instance to the input path.

saveImpl(*path*)

save() handles overwriting and then calls this method. Subclasses should override this method to implement the actual saving of the instance.

property **sc**

Returns the underlying *SparkContext*.

session(*sparkSession*)

Sets the Spark Session to use for saving.

property **sparkSession**

Returns the user-specified Spark Session or the default.

class pyspark.ml.util.**GeneralMLWriter** [source]

Utility class that can save ML instances in different formats.

New in version 2.4.0.

format(*source*) [source]

Specifies the format of ML export (e.g. "pmml", "internal", or the fully qualified class name for export).

overwrite()

Overwrites if the output path already exists.

save(*path*)

Save the ML instance to the input path.

saveImpl(*path*)

save() handles overwriting and then calls this method. Subclasses should override this method to implement the actual saving of the instance.

property **sc**

Returns the underlying *SparkContext*.

session(*sparkSession*)

Sets the Spark Session to use for saving/loading.

property **sparkSession**

Returns the user-specified Spark Session or the default.

class pyspark.ml.util.**HasTrainingSummary** [source]

Base class for models that provides Training summary. .. versionadded:: 3.0.0

property **hasSummary**

Indicates whether a training summary exists for this model instance.

New in version 2.1.0.

property **summary**

Gets summary of the model trained on the training set. An exception is thrown if no summary exists.

New in version 2.1.0.

class pyspark.ml.util.**Identifiable** [source]

Object with a unique ID.

uid = *None*

A unique id for the object.

class pyspark.ml.util.**JavaMLReadable** [source]

(Private) Mixin for instances that provide JavaMLReader.

classmethod **load**(*path*)

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

classmethod **read**() [source]

Returns an MLReader instance for this class.

class pyspark.ml.util.**JavaMLReader**(*clazz*) [source]

(Private) Specialization of **MLReader** for **JavaParams** types

load(*path*) [source]

Load the ML instance from the input path.

property **sc**

Returns the underlying *SparkContext*.

session(*sparkSession*) [source]

Sets the Spark Session to use for loading.

property **sparkSession**

Returns the user-specified Spark Session or the default.

class pyspark.ml.util.**JavaMLWritable** [source]

(Private) Mixin for ML instances that provide **JavaMLWriter**.

save(*path*)

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

write() [source]

Returns an MLWriter instance for this ML instance.

class pyspark.ml.util.**JavaMLWriter**(*instance*) [source]

(Private) Specialization of **MLWriter** for **JavaParams** types

option(*key*, *value*) [source]

overwrite() [source]

Overwrites if the output path already exists.

save(*path*) [source]

Save the ML instance to the input path.

saveImpl(*path*)

save() handles overwriting and then calls this method. Subclasses should override this method to implement the actual saving of the instance.

property **sc**

Returns the underlying *SparkContext*.

session(*sparkSession*) [source]

Sets the Spark Session to use for saving.

property **sparkSession**

Returns the user-specified Spark Session or the default.

class pyspark.ml.util.**MLReadable** [source]

Mixin for instances that provide **MLReader**.

New in version 2.0.0.

classmethod **load**(*path*) [source]

Reads an ML instance from the input path, a shortcut of *read().load(path)*.

classmethod **read**() [source]

Returns an MLReader instance for this class.

class pyspark.ml.util.**MLReader** [source]

Utility class that can load ML instances.

New in version 2.0.0.

load(*path*) [source]

Load the ML instance from the input path.

property **sc**

Returns the underlying *SparkContext*.

session(*sparkSession*)

Sets the Spark Session to use for saving/loading.

property **sparkSession**

Returns the user-specified Spark Session or the default.

class pyspark.ml.util.**MLWritable** [source]

Mixin for ML instances that provide **MLWriter**.

New in version 2.0.0.

save(*path*) [source]

Save this ML instance to the given path, a shortcut of 'write().save(path)'.

write() [source]

Returns an MLWriter instance for this ML instance.

class pyspark.ml.util.**MLWriter** [source]

Utility class that can save ML instances.

New in version 2.0.0.

overwrite() [source]

Overwrites if the output path already exists.

save(*path*) [source]

Save the ML instance to the input path.

saveImpl(*path*) [source]

save() handles overwriting and then calls this method. Subclasses should override this method to implement the actual saving of the instance.

property **sc**

Returns the underlying *SparkContext*.

session(*sparkSession*)

Sets the Spark Session to use for saving/loading.

property **sparkSession**

Returns the user-specified Spark Session or the default.