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`**

Abstract class for transformers that transform one dataset into another.

New in version 1.3.0.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`params`

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

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.

class `pyspark.ml.`**`Estimator`**

Abstract class for estimators that fit models to data.

New in version 1.3.0.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`params`

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

New in version 1.3.0.

class `pyspark.ml.`**`Model`**

Abstract class for models that are fitted by estimators.

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

params

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

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.

class `pyspark.ml.`**Pipeline**(*self*, *stages=None*)

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 there are no stages, the pipeline acts as an identity transformer.

New in version 1.3.0.

copy(*extra=None*)

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getStages()

Get pipeline stages.

New in version 1.3.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **load**(*path*)

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

params

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

New in version 1.3.0.

classmethod **`read`**()

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)*.

New in version 2.0.0.

`setParams`(*self*, *stages=None*)

Sets params for Pipeline.

New in version 1.3.0.

`setStages`(*value*)

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='pipeline stages')*

`write`()

Returns an MLWriter instance for this ML instance.

New in version 2.0.0.

class `pyspark.ml.`**`PipelineModel`**(*stages*)

Represents a compiled pipeline with transformers and fitted models.

New in version 1.3.0.

`copy`(*extra=None*)

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

classmethod **`read`**()

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)*.

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.

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.

`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

New in version 1.4.0.

`explainParam`(*param*) [source]

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

New in version 1.4.0.

`explainParams`() [source]

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

`getParam`(*paramName*) [source]

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*) [source]

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*) [source]

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

New in version 1.4.0.

`isDefined`(*param*) [source]

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

New in version 1.4.0.

`isSet`(*param*) [source]

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`params` [source]

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

New in version 1.3.0.

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

Convert a value to list of strings, 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**(*self*, *threshold=0.0*, *inputCol=None*, *outputCol=None*) [source]

Binarize a column of continuous features given a threshold.

```
>>> df = spark.createDataFrame([(0.5,)], ["values"])
>>> binarizer = Binarizer(threshold=1.0, inputCol="values", outputCol="features")
>>> 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
```

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getThreshold`() [source]

Gets the value of threshold or its default value.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setOutputCol`(*value*)

Sets the value of **`outputCol`**.

`setParams`(*self*, *threshold=0.0*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this Binarizer.

New in version 1.4.0.

`setThreshold`(*value*) [source]

Sets the value of **`threshold`**.

New in version 1.4.0.

`threshold` = *Param(parent='undefined', name='threshold', doc='threshold in binary classification prediction, in range [0, 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.

`write`()

Returns an MLWriter instance for this ML instance.

class `pyspark.ml.feature.`**`Bucketizer`**(*self*, *splits=None*, *inputCol=None*, *outputCol=None*) [source]

Maps a column of continuous features to a column of feature buckets.

```
>>> df = spark.createDataFrame([(0.1,), (0.4,), (1.2,), (1.5,)], ["values"])
>>> bucketizer = Bucketizer(splits=[-float("inf"), 0.5, 1.4, float("inf")],
...     inputCol="values", outputCol="buckets")
>>> bucketed = bucketizer.transform(df).collect()
>>> bucketed[0].buckets
0.0
>>> bucketed[1].buckets
0.0
>>> bucketed[2].buckets
1.0
>>> bucketed[3].buckets
2.0
>>> 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
```

New in version 1.4.0.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getSplits() [source]

Gets the value of threshold or its default value.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

params

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

New in version 1.3.0.

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)*.

setInputCol(*value*)

Sets the value of **inputCol**.

setOutputCol(*value*)

Sets the value of **outputCol**.

setParams(*self*, *splits=None*, *inputCol=None*, *outputCol=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.

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.')*

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**(*self*, *numTopFeatures=50*, *featuresCol="features"*, *outputCol=None*, *labelCol="label"*) [source]

Note: Experimental

Chi-Squared feature selection, which selects categorical features to use for predicting a categorical label.

```
>>> 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.transform(df).head().selectedFeatures
DenseVector([1.0])
>>> model.selectedFeatures
[3]
>>> 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.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

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.

getFeaturesCol()

Gets the value of featuresCol or its default value.

getLabelCol()

Gets the value of labelCol or its default value.

getNumTopFeatures() [source]

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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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 statistics value descending. If the number of features is < numTopFeatures, then this will select all features.')*

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

`params`

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

New in version 1.3.0.

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)*.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setNumTopFeatures`(*value*) [source]

Sets the value of **`numTopFeatures`**.

New in version 2.0.0.

`setOutputCol`(*value*)

Sets the value of **`outputCol`**.

`setParams`(*self*, *numTopFeatures=50*, *featuresCol="features"*, *outputCol=None*, *labelCol="labels"*) [source]

Sets params for this ChiSqSelector.

New in version 2.0.0.

`write`()

Returns an MLWriter instance for this ML instance.

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

Note: Experimental

Model fitted by **`ChiSqSelector`**.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **load**(*path*)

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

params

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

New in version 1.3.0.

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)*.

selectedFeatures [source]

List of indices to select (filter). Must be ordered asc.

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.`**`CountVectorizer`**(*self*, *minTF=1.0*, *minDF=1.0*, *vocabSize=1 << 18*, *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(inputCol="raw", outputCol="vectors")
>>> model = cv.fit(df)
>>> 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
```

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')*

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

`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.

`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.

`getMinDF`() [source]

Gets the value of minDF or its default value.

New in version 1.6.0.

`getMinTF`() [source]

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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getVocabSize`() [source]

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.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

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.')*

params

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

New in version 1.3.0.

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)*.

setBinary(*value*) [source]

Sets the value of `binary`.

New in version 2.0.0.

setInputCol(*value*)

Sets the value of `inputCol`.

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*)

Sets the value of **`outputCol`**.

`setParams`(*self*, *minTF=1.0*, *minDF=1.0*, *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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`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.

`vocabulary` [source]

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**(*self*, *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(inverse=False, inputCol="vec", outputCol="resultVec")
>>> 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.

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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setInverse`(*value*) [source]

Sets the value of **`inverse`**.

New in version 1.6.0.

`setOutputCol`(*value*)

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`**(*self*, *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(scalingVec=Vectors.dense([1.0, 2.0, 3.0]),
...     inputCol="values", outputCol="eprod")
>>> 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.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`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.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

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

`setInputCol`(*value*)

Sets the value of `inputCol`.

setOutputCol(*value*)

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.**HashingTF**(*self*, *numFeatures=1 << 18*, *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(numFeatures=10, inputCol="words", outputCol="features")
>>> hashingTF.transform(df).head().features
SparseVector(10, {0: 1.0, 1: 1.0, 2: 1.0})
>>> hashingTF.setParams(outputCol="freqs").transform(df).head().freqs
SparseVector(10, {0: 1.0, 1: 1.0, 2: 1.0})
>>> params = {hashingTF.numFeatures: 5, hashingTF.outputCol: "vector"}
>>> hashingTF.transform(df, params).head().vector
SparseVector(5, {0: 1.0, 1: 1.0, 2: 1.0})
>>> hashingTFPath = temp_path + "/hashing-tf"
>>> hashingTF.save(hashingTFPath)
>>> loadedHashingTF = HashingTF.load(hashingTFPath)
>>> loadedHashingTF.getNumFeatures() == hashingTF.getNumFeatures()
True
```

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.')*

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

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

`params`

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

New in version 1.3.0.

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)*.

`setBinary`(*value*) [source]

Sets the value of `binary`.

New in version 2.0.0.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setNumFeatures`(*value*)

Sets the value of **`numFeatures`**.

`setOutputCol`(*value*)

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`**(*self*, *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, inputCol="tf", outputCol="idf")
>>> model = idf.fit(df)
>>> model.idf
DenseVector([0.0, 0.0])
>>> 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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getInputCol`()

Gets the value of inputCol or its default value.

`getMinDocFreq`() [source]

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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setMinDocFreq`(*value*) [source]

Sets the value of **`minDocFreq`**.

New in version 1.4.0.

`setOutputCol`(*value*)

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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`idf` [source]

Returns the IDF vector.

New in version 2.0.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`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`**(*self*, *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.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setLabels`(*value*) [source]

Sets the value of **`labels`**.

New in version 1.6.0.

`setOutputCol`(*value*)

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.`**`MaxAbsScaler`**(*self*, *inputCol=None*, *outputCol=None*) [source]

Note: Experimental

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(inputCol="a", outputCol="scaled")
>>> model = maScaler.fit(df)
>>> model.transform(df).show()
+-----+------+
|    a|scaled|
+-----+------+
|[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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setOutputCol`(*value*)

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]

Note: Experimental

Model fitted by **`MaxAbsScaler`**.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **load**(*path*)

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

maxAbs [source]

Max Abs vector.

New in version 2.0.0.

params

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

New in version 1.3.0.

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)*.

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**(*self*, *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 that 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(inputCol="a", outputCol="scaled")
>>> model = mmScaler.fit(df)
>>> model.originalMin
DenseVector([0.0])
>>> model.originalMax
DenseVector([2.0])
>>> model.transform(df).show()
+-----+------+
|    a|scaled|
+-----+------+
|[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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

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.

`getInputCol`()

Gets the value of inputCol or its default value.

`getMax`() [source]

Gets the value of max or its default value.

New in version 1.6.0.

`getMin`() [source]

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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

setInputCol(*value*)

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*)

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.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`originalMax` [source]

Max value for each original column during fitting.

New in version 2.0.0.

`originalMin` [source]

Min value for each original column during fitting.

New in version 2.0.0.

`params`

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

New in version 1.3.0.

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)*.

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**(*self*, *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, inputCol="inputTokens", outputCol="nGrams")
>>> ngram.transform(df).head()
Row(inputTokens=[u'a', u'b', u'c', u'd', u'e'], nGrams=[u'a b', u'b c', u'c d', u'd e'])
>>> # Change n-gram length
>>> ngram.setParams(n=4).transform(df).head()
Row(inputTokens=[u'a', u'b', u'c', u'd', u'e'], nGrams=[u'a b c d', u'b c d e'])
>>> # Temporarily modify output column.
>>> ngram.transform(df, {ngram.outputCol: "output"}).head()
Row(inputTokens=[u'a', u'b', u'c', u'd', u'e'], output=[u'a b c d', u'b c d e'])
>>> ngram.transform(df).head()
Row(inputTokens=[u'a', u'b', u'c', u'd', u'e'], nGrams=[u'a b c d', u'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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setN`(*value*) [source]

Sets the value of **`n`**.

New in version 1.5.0.

`setOutputCol`(*value*)

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`**(*self*, *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, inputCol="dense", outputCol="features")
>>> 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.

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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`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.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`p` = *Param(parent='undefined', name='p', doc='the p norm value.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setOutputCol`(*value*)

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**(*self*, *includeFirst=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 that this is different from scikit-learn's OneHotEncoder, which keeps all categories. The output vectors are sparse.

See also: **StringIndexer** for converting categorical values into category indices

```
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed")
>>> model = stringIndexer.fit(stringIndDf)
>>> td = model.transform(stringIndDf)
>>> encoder = OneHotEncoder(inputCol="indexed", outputCol="features")
>>> encoder.transform(td).head().features
SparseVector(2, {0: 1.0})
>>> encoder.setParams(outputCol="freqs").transform(td).head().freqs
SparseVector(2, {0: 1.0})
>>> params = {encoder.dropLast: False, encoder.outputCol: "test"}
>>> encoder.transform(td, params).head().test
SparseVector(3, {0: 1.0})
>>> onehotEncoderPath = temp_path + "/onehot-encoder"
>>> encoder.save(onehotEncoderPath)
>>> loadedEncoder = OneHotEncoder.load(onehotEncoderPath)
>>> loadedEncoder.getDropLast() == encoder.getDropLast()
True
```

New in version 1.4.0.

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`getDropLast`() [source]

Gets the value of dropLast 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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setDropLast`(*value*) [source]

Sets the value of **`dropLast`**.

New in version 1.4.0.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setOutputCol`(*value*)

Sets the value of **`outputCol`**.

`setParams`(*self*, *dropLast=True*, *inputCol=None*, *outputCol=None*) [source]

Sets params for this OneHotEncoder.

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.`**PCA**(*self*, *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", outputCol="pca_features")
>>> model = pca.fit(df)
>>> model.transform(df).collect()[0].pca_features
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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

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.

getInputCol()

Gets the value of inputCol or its default value.

getK() [source]

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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

params

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

New in version 1.3.0.

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)*.

setInputCol(*value*)

Sets the value of `inputCol`.

setK(*value*) [source]

Sets the value of `k`.

New in version 1.5.0.

setOutputCol(*value*)

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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`explainedVariance` [source]

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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **load**(*path*)

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

params

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

New in version 1.3.0.

`pc` [source]

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)*.

`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`**(*self*, *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, inputCol="dense", outputCol="expanded")
>>> 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.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

params

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

New in version 1.3.0.

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)*.

setDegree(*value*) [source]

Sets the value of **degree**.

New in version 1.4.0.

setInputCol(*value*)

Sets the value of **inputCol**.

setOutputCol(*value*)

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`**(*self*, *numBuckets=2*, *inputCol=None*, *outputCol=None*, *relativeError=0.001*) [source]

Note: Experimental

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. 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.

```
>>> df = spark.createDataFrame([(0.1,), (0.4,), (1.2,), (1.5,)], ["values"])
>>> qds = QuantileDiscretizer(numBuckets=2,
...     inputCol="values", outputCol="buckets", relativeError=0.01)
>>> qds.getRelativeError()
0.01
>>> bucketizer = qds.fit(df)
>>> splits = bucketizer.getSplits()
>>> splits[0]
-inf
>>> print("%2.1f" % round(splits[1], 1))
```

```
0.4
>>> bucketed = bucketizer.transform(df).head()
>>> bucketed.buckets
0.0
>>> quantileDiscretizerPath = temp_path + "/quantile-discretizer"
>>> qds.save(quantileDiscretizerPath)
>>> loadedQds = QuantileDiscretizer.load(quantileDiscretizerPath)
>>> loadedQds.getNumBuckets() == qds.getNumBuckets()
True
```

New in version 2.0.0.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getInputCol`()

Gets the value of inputCol or its default value.

`getNumBuckets`() [source]

Gets the value of numBuckets 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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getRelativeError`() [source]

Gets the value of relativeError or its default value.

New in version 2.0.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

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

`params`

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

New in version 1.3.0.

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 used to generate buckets. Must be in the range [0, 1].')*

save(*path*)

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

setInputCol(*value*)

Sets the value of **inputCol**.

setNumBuckets(*value*) [source]

Sets the value of **numBuckets**.

New in version 2.0.0.

setOutputCol(*value*)

Sets the value of **outputCol**.

setParams(*self*, *numBuckets=2*, *inputCol=None*, *outputCol=None*, *relativeError=0.001*) [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.**RegexTokenizer**(*self*, *minTokenLength=1*, *gaps=True*, *pattern="s+"*, *inputCol=None*, *outputCol=None*, *toLowercase=True*)

A regex based tokenizer that extracts tokens either by using the provided regex pattern (in Java dialect) to split the text (default) or repeatedly [source] 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(inputCol="text", outputCol="words")
>>> reTokenizer.transform(df).head()
Row(text=u'A B  c', words=[u'a', u'b', u'c'])
>>> # Change a parameter.
>>> reTokenizer.setParams(outputCol="tokens").transform(df).head()
Row(text=u'A B  c', tokens=[u'a', u'b', u'c'])
>>> # Temporarily modify a parameter.
>>> reTokenizer.transform(df, {reTokenizer.outputCol: "words"}).head()
Row(text=u'A B  c', words=[u'a', u'b', u'c'])
>>> reTokenizer.transform(df).head()
Row(text=u'A B  c', tokens=[u'a', u'b', u'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.

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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

`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)*.

`setGaps`(*value*) [source]

Sets the value of **`gaps`**.

New in version 1.4.0.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setMinTokenLength`(*value*) [source]

Sets the value of **`minTokenLength`**.

New in version 1.4.0.

`setOutputCol`(*value*)

Sets the value of **`outputCol`**.

`setParams`(*self*, *minTokenLength=1*, *gaps=True*, *pattern="s+"*, *inputCol=None*, *outputCol=None*, *toLowercase=True*) [source]

Sets params for this RegexTokenizer.

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`**(*self*, *formula=None*, *featuresCol="features"*, *labelCol="label"*) [source]

Note: Experimental

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.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
>>> 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.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`formula` = *Param(parent='undefined', name='formula', doc='R model formula')*

`getFeaturesCol`()

Gets the value of featuresCol or its default value.

`getFormula`() [source]

Gets the value of `formula`.

New in version 1.5.0.

`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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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)*.

params

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

New in version 1.3.0.

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)*.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setFormula`(*value*) [source]

Sets the value of **`formula`**.

New in version 1.5.0.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setParams`(*self*, *formula=None*, *featuresCol="features"*, *labelCol="label"*) [source]

Sets params for RFormula.

New in version 1.5.0.

`write`()

Returns an MLWriter instance for this ML instance.

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

Note: Experimental

Model fitted by **`RFormula`**. Fitting is required to determine the factor levels of formula terms.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`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**(*self*, *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.

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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`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.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`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**(*self*, *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(inputCol="a", outputCol="scaled")
>>> model = standardScaler.fit(df)
>>> model.mean
DenseVector([1.0])
>>> model.std
DenseVector([1.4142])
>>> model.transform(df).collect()[1].scaled
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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

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.

`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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getWithMean`() [source]

Gets the value of withMean or its default value.

New in version 1.4.0.

`getWithStd`() [source]

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.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setOutputCol`(*value*)

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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **load**(*path*)

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

mean [source]

Mean of the StandardScalerModel.

New in version 2.0.0.

params

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

New in version 1.3.0.

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)*.

std [source]

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.

`write`()

Returns an MLWriter instance for this ML instance.

class `pyspark.ml.feature.`**`StopWordsRemover`**(*self*, *inputCol=None*, *outputCol=None*, *stopWords=None*, *caseSensitive=false*) [source]

A feature transformer that filters out stop words from input. Note: null values from input array are preserved unless adding null to stopWords explicitly.

```
>>> df = spark.createDataFrame([(["a", "b", "c"],)], ["text"])
>>> remover = StopWordsRemover(inputCol="text", outputCol="words", stopWords=["b"])
>>> 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
```

New in version 1.6.0.

`caseSensitive` = *Param(parent='undefined', name='caseSensitive', doc='whether to do a case sensitive comparison over the stop words')*

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`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.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.

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

`params`

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

New in version 1.3.0.

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)*.

`setCaseSensitive`(*value*) [source]

Sets the value of **`caseSensitive`**.

New in version 1.6.0.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setOutputCol`(*value*)

Sets the value of **`outputCol`**.

`setParams`(*self*, *inputCol=None*, *outputCol=None*, *stopWords=None*, *caseSensitive=false*) [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**(*self*, *inputCol=None*, *outputCol=None*, *handleInvalid="error"*) [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), ordered by label frequencies. So the most frequent label gets index 0.

```
>>> stringIndexer = StringIndexer(inputCol="label", outputCol="indexed", handleInvalid='error')
>>> 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, 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
```

New in version 1.4.0.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

`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.

`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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`handleInvalid` *= Param(parent='undefined', name='handleInvalid', doc='how to handle invalid entries. Options are skip (which will filter out rows with bad values), or error (which will throw an error). More options may be added later.')*

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

params

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

New in version 1.3.0.

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)*.

setHandleInvalid(*value*)

Sets the value of **`handleInvalid`**.

setInputCol(*value*)

Sets the value of **`inputCol`**.

setOutputCol(*value*)

Sets the value of **`outputCol`**.

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

Sets params for this StringIndexer.

New in version 1.4.0.

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.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`labels` [source]

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)*.

`params`

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

New in version 1.3.0.

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)*.

`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`**(*self*, *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(inputCol="text", outputCol="words")
>>> tokenizer.transform(df).head()
Row(text=u'a b c', words=[u'a', u'b', u'c'])
>>> # Change a parameter.
>>> tokenizer.setParams(outputCol="tokens").transform(df).head()
Row(text=u'a b c', tokens=[u'a', u'b', u'c'])
>>> # Temporarily modify a parameter.
>>> tokenizer.transform(df, {tokenizer.outputCol: "words"}).head()
Row(text=u'a b c', words=[u'a', u'b', u'c'])
>>> tokenizer.transform(df).head()
Row(text=u'a b c', tokens=[u'a', u'b', u'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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

params

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setOutputCol`(*value*)

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

A feature transformer that merges multiple columns into a vector column.

```
>>> df = spark.createDataFrame([(1, 0, 3)], ["a", "b", "c"])
>>> vecAssembler = VectorAssembler(inputCols=["a", "b", "c"], outputCol="features")
>>> 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
```

New in version 1.4.0.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCols`(*value*)

Sets the value of **`inputCols`**.

`setOutputCol`(*value*)

Sets the value of **`outputCol`**.

`setParams`(*self*, *inputCols=None*, *outputCol=None*) [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`**(*self*, *maxCategories=20*, *inputCol=None*, *outputCol=None*) [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.
- Add option for allowing unknown categories.

```
>>> 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", outputCol="indexed")
>>> model = indexer.fit(df)
>>> model.transform(df).head().indexed
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
```

New in version 1.4.0.

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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getInputCol`()

Gets the value of inputCol or its default value.

`getMaxCategories`() [source]

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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setMaxCategories`(*value*) [source]

Sets the value of **`maxCategories`**.

New in version 1.4.0.

`setOutputCol`(*value*)

Sets the value of **`outputCol`**.

`setParams`(*self*, *maxCategories=20*, *inputCol=None*, *outputCol=None*) [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.

`categoryMaps` [source]

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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`numFeatures` [source]

Number of features, i.e., length of Vectors which this transforms.

New in version 1.4.0.

`params`

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

New in version 1.3.0.

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)*.

`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`**(*self*, *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(inputCol="features", outputCol="sliced", indices=[1, 4])
>>> 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.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

`getOutputCol`()

Gets the value of outputCol or its default value.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

`params`

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

New in version 1.3.0.

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)*.

`setIndices`(*value*) [source]

Sets the value of **`indices`**.

New in version 1.6.0.

`setInputCol`(*value*)

Sets the value of **`inputCol`**.

`setNames`(*value*) [source]

Sets the value of **`names`**.

New in version 1.6.0.

`setOutputCol`(*value*)

Sets the value of **`outputCol`**.

`setParams`(**args*, ***kwargs*) [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`**(*self*, *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")
>>> model = word2Vec.fit(doc)
>>> model.getVectors().show()
+----+--------------------+
|word|              vector|
+----+--------------------+
|   a|[0.09461779892444...|
|   b|[1.15474212169647...|
|   c|[-0.3794820010662...|
+----+--------------------+
...
>>> from pyspark.sql.functions import format_number as fmt
>>> model.findSynonyms("a", 2).select("word", fmt("similarity", 5).alias("similarity")).show()
+----+----------+
|word|similarity|
+----+----------+
|   b|   0.25053|
|   c|  -0.69805|
+----+----------+
...
>>> model.transform(doc).head().model
DenseVector([0.5524, -0.4995, -0.3599, 0.0241, 0.3461])
>>> 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.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getInputCol`()

Gets the value of inputCol or its default value.

`getMaxIter`()

Gets the value of maxIter or its default value.

`getMaxSentenceLength`() [source]

Gets the value of maxSentenceLength or its default value.

New in version 2.0.0.

`getMinCount`() [source]

Gets the value of minCount or its default value.

New in version 1.4.0.

`getNumPartitions`() [source]

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.

New in version 1.4.0.

getOutputCol()

Gets the value of outputCol or its default value.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getSeed()

Gets the value of seed or its default value.

getStepSize()

Gets the value of stepSize or its default value.

getVectorSize() [source]

Gets the value of vectorSize or its default value.

New in version 1.4.0.

getWindowSize() [source]

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.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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.')*

params

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

New in version 1.3.0.

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)*.

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

setInputCol(*value*)

Sets the value of **inputCol**.

setMaxIter(*value*)

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*)

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*)

Sets the value of **seed**.

setStepSize(*value*)

Sets the value of **stepSize**.

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.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

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.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

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.

pyspark.ml.classification module

class `pyspark.ml.classification.`**LogisticRegression**(*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*) [source]

Logistic regression. Currently, this class only supports binary classification.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = sc.parallelize([
...     Row(label=1.0, weight=2.0, features=Vectors.dense(1.0)),
...     Row(label=0.0, weight=2.0, features=Vectors.sparse(1, [], []))]).toDF()
>>> lr = LogisticRegression(maxIter=5, regParam=0.01, weightCol="weight")
>>> model = lr.fit(df)
>>> model.coefficients
DenseVector([5.5...])
>>> model.intercept
-2.68...
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0))]).toDF()
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> result.probability
DenseVector([0.99..., 0.00...])
```

```
>>> result.rawPrediction
DenseVector([8.22..., -8.22...])
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(1, [0], [1.0]))]).toDF()
>>> model.transform(test1).head().prediction
1.0
>>> lr.setParams("vector")
Traceback (most recent call last):
    ...
TypeError: Method setParams forces keyword arguments.
>>> lr_path = temp_path + "/lr"
>>> lr.save(lr_path)
>>> lr2 = LogisticRegression.load(lr_path)
>>> lr2.getMaxIter()
5
>>> model_path = temp_path + "/lr_model"
>>> model.save(model_path)
>>> model2 = LogisticRegressionModel.load(model_path)
>>> model.coefficients[0] == model2.coefficients[0]
True
>>> model.intercept == model2.intercept
True
```

New in version 1.3.0.

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

New in version 1.4.0.

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.')*

`getElasticNetParam`()

Gets the value of elasticNetParam 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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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() [source]

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() [source]

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.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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).')*

`params`

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

New in version 1.3.0.

`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)*.

`setElasticNetParam`(*value*)

Sets the value of **`elasticNetParam`**.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setFitIntercept`(*value*)

Sets the value of **`fitIntercept`**.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setMaxIter`(*value*)

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*, *threshold=0.5*, *thresholds=None*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *standardization=True*, *weightCol=None*) [source]

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`**.

`setProbabilityCol`(*value*)

Sets the value of **`probabilityCol`**.

`setRawPredictionCol`(*value*)

Sets the value of **`rawPredictionCol`**.

`setRegParam`(*value*)

Sets the value of **`regParam`**.

`setStandardization`(*value*)

Sets the value of **`standardization`**.

`setThreshold`(*value*) [source]

Sets the value of **`threshold`**. Clears value of **`thresholds`** if it has been set.

New in version 1.4.0.

setThresholds(*value*) [source]

Sets the value of **thresholds**. Clears value of **threshold** if it has been set.

New in version 1.5.0.

setTol(*value*)

Sets the value of **tol**.

setWeightCol(*value*)

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. The class with largest value p/t is predicted, where p is the original probability of that class and t is the class' threshold.")*

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.**LogisticRegressionModel**(*java_model=None*) [source]

Model fitted by LogisticRegression.

New in version 1.3.0.

coefficients [source]

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

hasSummary [source]

Indicates whether a training summary exists for this model instance.

New in version 2.0.0.

intercept [source]

Model intercept.

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`summary` [source]

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.

`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.classification.`**`LogisticRegressionSummary`**(*java_obj=None*) [source]

Note: Experimental

Abstraction for Logistic Regression Results for a given model.

New in version 2.0.0.

`featuresCol` [source]

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

`labelCol` [source]

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

`predictions` [source]

Dataframe outputted by the model’s *transform* method.

New in version 2.0.0.

`probabilityCol` [source]

Field in “predictions” which gives the probability of each class as a vector.

New in version 2.0.0.

class `pyspark.ml.classification.`**`LogisticRegressionTrainingSummary`**(*java_obj=None*) [source]

Note: Experimental

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.

`featuresCol`

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

`labelCol`

Field in "predictions" which gives the true label of each instance.

New in version 2.0.0.

`objectiveHistory` [source]

Objective function (scaled loss + regularization) at each iteration.

New in version 2.0.0.

`predictions`

Dataframe outputted by the model's *transform* method.

New in version 2.0.0.

`probabilityCol`

Field in "predictions" which gives the probability of each class as a vector.

New in version 2.0.0.

`totalIterations` [source]

Number of training iterations until termination.

New in version 2.0.0.

class `pyspark.ml.classification.`**`BinaryLogisticRegressionSummary`**(*java_obj=None*) [source]

Note: Experimental

Binary Logistic regression results for a given model.

New in version 2.0.0.

`areaUnderROC` [source]

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.

`fMeasureByThreshold` [source]

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.

`featuresCol`

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

`labelCol`

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

`pr` [source]

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.

`precisionByThreshold` [source]

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.

`predictions`

Dataframe outputted by the model's *transform* method.

New in version 2.0.0.

`probabilityCol`

Field in "predictions" which gives the probability of each class as a vector.

New in version 2.0.0.

`recallByThreshold` [source]

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.

`roc` [source]

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.

class `pyspark.ml.classification.`**`BinaryLogisticRegressionTrainingSummary`**(*java_obj=None*) [source]

Note: Experimental

Binary Logistic regression training results for a given model.

New in version 2.0.0.

`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.

`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.

`featuresCol`

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

`labelCol`

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

`objectiveHistory`

Objective function (scaled loss + regularization) at each iteration.

New in version 2.0.0.

`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.

`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.

`predictions`

Dataframe outputted by the model's *transform* method.

New in version 2.0.0.

`probabilityCol`

Field in "predictions" which gives the probability of each class as a vector.

New in version 2.0.0.

`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.

`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.

`totalIterations`

Number of training iterations until termination.

New in version 2.0.0.

class `pyspark.ml.classification.`**`DecisionTreeClassifier`**(*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*) [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")
>>> model = dt.fit(td)
>>> model.numNodes
3
>>> model.depth
1
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> print(model.toDebugString)
DecisionTreeClassificationModel (uid=...) of depth 1 with 3 nodes...
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> result.probability
DenseVector([1.0, 0.0])
>>> result.rawPrediction
DenseVector([1.0, 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
```

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.')*

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

`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.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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)*.

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.')*

params

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

New in version 1.3.0.

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.')*

setCacheNodeIds(*value*)

Sets the value of `cacheNodeIds`.

setCheckpointInterval(*value*)

Sets the value of `checkpointInterval`.

setFeaturesCol(*value*)

Sets the value of `featuresCol`.

setImpurity(*value*)

Sets the value of `impurity`.

New in version 1.6.0.

`setLabelCol`(*value*)

Sets the value of `labelCol`.

`setMaxBins`(*value*)

Sets the value of `maxBins`.

`setMaxDepth`(*value*)

Sets the value of `maxDepth`.

`setMaxMemoryInMB`(*value*)

Sets the value of `maxMemoryInMB`.

`setMinInfoGain`(*value*)

Sets the value of `minInfoGain`.

`setMinInstancesPerNode`(*value*)

Sets the value of `minInstancesPerNode`.

`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*) [source]

Sets params for the DecisionTreeClassifier.

New in version 1.4.0.

`setPredictionCol`(*value*)

Sets the value of `predictionCol`.

`setProbabilityCol`(*value*)

Sets the value of `probabilityCol`.

`setRawPredictionCol`(*value*)

Sets the value of `rawPredictionCol`.

`setSeed`(*value*)

Sets the value of **`seed`**.

`supportedImpurities` = *['entropy', 'gini']*

`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.

`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 model with extra params. So both the Python wrapper and the Java model get copied.

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

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`featureImportances` [source]

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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`numNodes`

Return number of nodes of the decision tree.

New in version 1.5.0.

`params`

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

New in version 1.3.0.

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)*.

`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.

`write`()

Returns an MLWriter instance for this ML instance.

class `pyspark.ml.classification.`**`GBTClassifier`**(*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*) [source]

Gradient-Boosted Trees (GBTs) learning algorithm for classification. It supports binary labels, as well as both continuous and categorical features. Note: Multiclass labels are not currently supported.

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

```
>>> 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)
>>> model = gbt.fit(td)
>>> 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.transform(test0).head().prediction
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 (uid=...)...with 5 trees...
>>> 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 (uid=...) of depth..., DecisionTreeRegressionModel...]
```

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.')*

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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

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.

getLabelCol()

Gets the value of labelCol or its default value.

getLossType() [source]

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.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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)*.

`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.')*

`params`

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

New in version 1.3.0.

`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.')*

`setCacheNodeIds`(*value*)

Sets the value of **`cacheNodeIds`**.

`setCheckpointInterval`(*value*)

Sets the value of **`checkpointInterval`**.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setLossType`(*value*) [source]

Sets the value of **`lossType`**.

New in version 1.4.0.

`setMaxBins`(*value*)

Sets the value of **`maxBins`**.

`setMaxDepth`(*value*)

Sets the value of **`maxDepth`**.

`setMaxIter`(*value*)

Sets the value of **`maxIter`**.

`setMaxMemoryInMB`(*value*)

Sets the value of **`maxMemoryInMB`**.

`setMinInfoGain`(*value*)

Sets the value of **`minInfoGain`**.

`setMinInstancesPerNode`(*value*)

Sets the value of **`minInstancesPerNode`**.

`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*)

Sets params for Gradient Boosted Tree Classification. [source]

New in version 1.4.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`setStepSize`(*value*)

Sets the value of **`stepSize`**.

`setSubsamplingRate`(*value*)

Sets the value of **`subsamplingRate`**.

New in version 1.4.0.

`stepSize` = *Param(parent='undefined', name='stepSize', doc='Step size to be used for each iteration of optimization (>= 0).')*

`subsamplingRate` = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in* *range (0, 1].')*

supportedLossTypes = *['logistic']*

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.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

`featureImportances` [source]

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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`toDebugString`

Full description of model.

New in version 2.0.0.

`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.

treeWeights

Return the weights for each tree

New in version 1.5.0.

trees [source]

Trees in this ensemble. Warning: These have null parent Estimators.

New in version 2.0.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**RandomForestClassifier**(*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"*, *numTrees=20*, *featureSubsetStrategy="auto"*, *seed=None*) [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)
>>> model = rf.fit(td)
>>> 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"])
>>> result = model.transform(test0).head()
>>> result.prediction
0.0
>>> numpy.argmax(result.probability)
0
>>> numpy.argmax(result.rawPrediction)
0
>>> test1 = spark.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
>>> model.trees
[DecisionTreeClassificationModel (uid=...) of 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.')*

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`featureSubsetStrategy` = *Param(parent='undefined', name='featureSubsetStrategy', doc='The number of features to consider for splits at each tree node. Supported options: auto, all, onethird, sqrt, log2 (0.0-1.0], [1-n].')*

`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.

`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.

`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.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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)*.

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.')*

numTrees = *Param(parent='undefined', name='numTrees', doc='Number of trees to train (>= 1).')*

params

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

New in version 1.3.0.

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.')*

setCacheNodeIds(*value*)

Sets the value of `cacheNodeIds`.

setCheckpointInterval(*value*)

Sets the value of `checkpointInterval`.

setFeatureSubsetStrategy(*value*)

Sets the value of `featureSubsetStrategy`.

New in version 1.4.0.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setImpurity`(*value*)

Sets the value of **`impurity`**.

New in version 1.6.0.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setMaxBins`(*value*)

Sets the value of **`maxBins`**.

`setMaxDepth`(*value*)

Sets the value of **`maxDepth`**.

`setMaxMemoryInMB`(*value*)

Sets the value of **`maxMemoryInMB`**.

`setMinInfoGain`(*value*)

Sets the value of **`minInfoGain`**.

`setMinInstancesPerNode`(*value*)

Sets the value of **`minInstancesPerNode`**.

`setNumTrees`(*value*)

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"*) [source]

Sets params for linear classification.

New in version 1.4.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`setProbabilityCol`(*value*)

Sets the value of **`probabilityCol`**.

`setRawPredictionCol`(*value*)

Sets the value of **`rawPredictionCol`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`setSubsamplingRate`(*value*)

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` = *['entropy', 'gini']*

`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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`featureImportances` [source]

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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`toDebugString`

Full description of model.

New in version 2.0.0.

`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.

`treeWeights`

Return the weights for each tree

New in version 1.5.0.

trees [source]

Trees in this ensemble. Warning: These have null parent Estimators.

New in version 2.0.0.

write()

Returns an MLWriter instance for this ML instance.

class pyspark.ml.classification.**NaiveBayes**(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *smoothing=1.0*, *modelType="multinomial"*, *thresholds=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 must be nonnegative.

```
>>> from pyspark.sql import Row
>>> from pyspark.ml.linalg import Vectors
>>> df = spark.createDataFrame([
...     Row(label=0.0, features=Vectors.dense([0.0, 0.0])),
...     Row(label=0.0, features=Vectors.dense([0.0, 1.0])),
...     Row(label=1.0, features=Vectors.dense([1.0, 0.0]))])
>>> nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
>>> model = nb.fit(df)
>>> model.pi
DenseVector([-0.51..., -0.91...])
>>> model.theta
DenseMatrix(2, 2, [-1.09..., -0.40..., -0.40..., -1.09...], 1)
>>> test0 = sc.parallelize([Row(features=Vectors.dense([1.0, 0.0]))]).toDF()
>>> result = model.transform(test0).head()
>>> result.prediction
1.0
>>> result.probability
DenseVector([0.42..., 0.57...])
>>> result.rawPrediction
DenseVector([-1.60..., -1.32...])
>>> 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
```

New in version 1.5.0.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

`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.

`getFeaturesCol`()

Gets the value of featuresCol or its default value.

`getLabelCol`()

Gets the value of labelCol or its default value.

`getModelType`() [source]

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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() [source]

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.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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) and bernoulli.')*

params

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

New in version 1.3.0.

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)*.

setFeaturesCol(*value*)

Sets the value of `featuresCol`.

setLabelCol(*value*)

Sets the value of **labelCol**.

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*) [source]

Sets params for Naive Bayes.

New in version 1.5.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

setProbabilityCol(*value*)

Sets the value of **probabilityCol**.

setRawPredictionCol(*value*)

Sets the value of **rawPredictionCol**.

setSmoothing(*value*) [source]

Sets the value of **smoothing**.

New in version 1.5.0.

setThresholds(*value*)

Sets the value of **thresholds**.

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. The class with largest value p/t is predicted, where p is the original* *probability of that class and t is the class' threshold.")*

`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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **load**(*path*)

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

params

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

New in version 1.3.0.

`pi` [source]

log of class priors.

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)*.

`theta` [source]

log of class conditional probabilities.

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.classification.`**`MultilayerPerceptronClassifier`**(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxIter=100*, *tol=1e-4*, *seed=None*, *layers=None*, *blockSize=128*, *stepSize=0.03*, *solver="l-bfgs"*, *initialWeights=None*) [source]

Note: Experimental

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(maxIter=100, layers=[2, 2, 2], blockSize=1, seed=123)
>>> model = mlp.fit(df)
>>> 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.transform(testDF).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.')*

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getBlockSize`() [source]

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`() [source]

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`() [source]

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getPredictionCol`()

Gets the value of predictionCol or its default value.

`getSeed`()

Gets the value of seed or its default value.

`getSolver`() [source]

Gets the value of solver or its default value.

New in version 2.0.0.

`getStepSize`() [source]

Gets the value of stepSize or its default value.

New in version 2.0.0.

`getTol`()

Gets the value of tol or its default value.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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).')*

params

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

New in version 1.3.0.

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.')*

setBlockSize(*value*) [source]

Sets the value of **`blockSize`**.

New in version 1.6.0.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setInitialWeights`(*value*) [source]

Sets the value of **`initialWeights`**.

New in version 2.0.0.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setLayers`(*value*) [source]

Sets the value of **`layers`**.

New in version 1.6.0.

`setMaxIter`(*value*)

Sets the value of **`maxIter`**.

`setParams`(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxIter=100*, *tol=1e-4*, *seed=None*, *layers=None*, *blockSize=128*, *stepSize=0.03*, *solver="l-bfgs"*, *initialWeights=None*) [source]

Sets params for MultilayerPerceptronClassifier.

New in version 1.6.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`setSolver`(*value*) [source]

Sets the value of **`solver`**.

New in version 2.0.0.

`setStepSize`(*value*) [source]

Sets the value of **`stepSize`**.

New in version 2.0.0.

`setTol`(*value*)

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).')*

`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]

Note: Experimental

Model fitted by MultilayerPerceptronClassifier.

New in version 1.6.0.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

layers [source]

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)*.

params

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

New in version 1.3.0.

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)*.

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.

`weights` [source]

vector of initial weights for the model that consists of the weights of layers.

New in version 2.0.0.

`write`()

Returns an MLWriter instance for this ML instance.

class `pyspark.ml.classification.`**`OneVsRest`**(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *classifier=None*) [source]

Note: Experimental

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
>>> df = sc.parallelize([
...     Row(label=0.0, features=Vectors.dense(1.0, 0.8)),
...     Row(label=1.0, features=Vectors.sparse(2, [], [])),
...     Row(label=2.0, features=Vectors.dense(0.5, 0.5))]).toDF()
>>> lr = LogisticRegression(maxIter=5, regParam=0.01)
>>> ovr = OneVsRest(classifier=lr)
>>> model = ovr.fit(df)
>>> [x.coefficients for x in model.models]
[DenseVector([3.3925, 1.8785]), DenseVector([-4.3016, -6.3163]), DenseVector([-4.5855, 6.1785])]
>>> [x.intercept for x in model.models]
[-3.6474708290602034, 2.5507881951814495, -1.1016513228162115]
>>> test0 = sc.parallelize([Row(features=Vectors.dense(-1.0, 0.0))]).toDF()
>>> model.transform(test0).head().prediction
1.0
>>> test1 = sc.parallelize([Row(features=Vectors.sparse(2, [0], [1.0]))]).toDF()
>>> model.transform(test1).head().prediction
0.0
>>> test2 = sc.parallelize([Row(features=Vectors.dense(0.5, 0.4))]).toDF()
>>> model.transform(test2).head().prediction
2.0
```

New in version 2.0.0.

`classifier` = *Param(parent='undefined', name='classifier', doc='base binary classifier')*

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getPredictionCol`()

Gets the value of predictionCol or its default value.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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)*.

`params`

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

New in version 1.3.0.

`predictionCol` = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **`read`**() [source]

Returns an MLReader instance for this class.

New in version 2.0.0.

`save`(*path*) [source]

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

New in version 2.0.0.

`setClassifier`(*value*)

Sets the value of **`classifier`**.

> **Note:** Only LogisticRegression and NaiveBayes are supported now.

New in version 2.0.0.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setParams`(**args*, ***kwargs*) [source]

setParams(self, featuresCol=None, labelCol=None, predictionCol=None, classifier=None): Sets params for OneVsRest.

New in version 2.0.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`write`() [source]

Returns an MLWriter instance for this ML instance.

New in version 2.0.0.

class `pyspark.ml.classification.`**`OneVsRestModel`**(*models*) [source]

> **Note:** Experimental

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')*

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getPredictionCol()

Gets the value of predictionCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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)*.

`params`

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

New in version 1.3.0.

`predictionCol` = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

classmethod **`read`**() [source]

Returns an MLReader instance for this class.

New in version 2.0.0.

`save`(*path*) [source]

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

New in version 2.0.0.

`setClassifier`(*value*)

Sets the value of **`classifier`**.

Note: Only LogisticRegression and NaiveBayes are supported now.

New in version 2.0.0.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`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.clustering module

class `pyspark.ml.clustering.`**`BisectingKMeans`**(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *maxIter=20*, *seed=None*, *k=4*, *minDivisibleClusterSize=1.0*) [source]

Note: Experimental

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)
>>> model = bkm.fit(df)
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> model.computeCost(df)
2.000...
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[0].prediction == rows[1].prediction
True
>>> rows[2].prediction == rows[3].prediction
True
>>> bkm_path = temp_path + "/bkm"
>>> bkm.save(bkm_path)
>>> bkm2 = BisectingKMeans.load(bkm_path)
>>> bkm2.getK()
2
>>> model_path = temp_path + "/bkm_model"
>>> model.save(model_path)
>>> model2 = BisectingKMeansModel.load(model_path)
>>> 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.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getFeaturesCol`()

Gets the value of featuresCol or its default value.

`getK`() [source]

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() [source]

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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.')*

`params`

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

New in version 1.3.0.

`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.')*

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setK`(*value*) [source]

Sets the value of **`k`**.

New in version 2.0.0.

`setMaxIter`(*value*)

Sets the value of **`maxIter`**.

`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*) [source]

Sets params for BisectingKMeans.

New in version 2.0.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`write`()

Returns an MLWriter instance for this ML instance.

class `pyspark.ml.clustering.`**`BisectingKMeansModel`**(*java_model=None*) [source]

Note: Experimental

Model fitted by BisectingKMeans.

New in version 2.0.0.

`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.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

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.**KMeans**(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *k=2*, *initMode="k-means||"*, *initSteps=5*, *tol=1e-4*, *maxIter=20*, *seed=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]),), (Vectors.dense([1.0, 1.0]),),
...         (Vectors.dense([9.0, 8.0]),), (Vectors.dense([8.0, 9.0]),)]
>>> df = spark.createDataFrame(data, ["features"])
>>> kmeans = KMeans(k=2, seed=1)
>>> model = kmeans.fit(df)
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> model.computeCost(df)
2.000...
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[0].prediction == rows[1].prediction
True
>>> rows[2].prediction == rows[3].prediction
True
>>> 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)
>>> 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.

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getFeaturesCol`()

Gets the value of featuresCol or its default value.

`getInitMode`() [source]

Gets the value of *initMode*

New in version 1.5.0.

`getInitSteps`() [source]

Gets the value of *initSteps*

New in version 1.5.0.

`getK`() [source]

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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).')*

`params`

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

New in version 1.3.0.

`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.')*

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`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*)

Sets the value of **`maxIter`**.

`setParams`(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *k=2*, *initMode="k-means||"*, *initSteps=5*, *tol=1e-4*, *maxIter=20*, *seed=None*) [source]

Sets params for KMeans.

New in version 1.5.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`setTol`(*value*)

Sets the value of **`tol`**.

`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.`**`KMeansModel`**(*java_model=None*) [source]

Model fitted by KMeans.

New in version 1.5.0.

`clusterCenters`() [source]

Get the cluster centers, represented as a list of NumPy arrays.

New in version 1.5.0.

`computeCost`(*dataset*) [source]

Return the K-means cost (sum of squared distances of points to their nearest center) for this model on the given data.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **load**(*path*)

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

params

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

New in version 1.3.0.

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)*.

`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.`**`GaussianMixture`**(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *k=2*, *probabilityCol="probability"*, *tol=0.01*, *maxIter=100*, *seed=None*) [source]

Note: Experimental

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,
...                      maxIter=10, seed=10)
>>> model = gm.fit(df)
>>> weights = model.weights
>>> len(weights)
3
>>> model.gaussiansDF.show()
+--------------------+--------------------+
|                mean|                 cov|
+--------------------+--------------------+
|[-0.0550000000000...|0.002025000000000...|
|[0.82499999999999...|0.005625000000000...|
|[-0.87,-0.7200000...|0.001600000000000...|
+--------------------+--------------------+
...
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[4].prediction == rows[5].prediction
True
>>> rows[2].prediction == rows[3].prediction
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.weights == model.weights
True
>>> model2.gaussiansDF.show()
+--------------------+--------------------+
|                mean|                 cov|
+--------------------+--------------------+
|[-0.0550000000000...|0.002025000000000...|
|[0.82499999999999...|0.005625000000000...|
|[-0.87,-0.7200000...|0.001600000000000...|
+--------------------+--------------------+
...
```

New in version 2.0.0.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

`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.

`getFeaturesCol`()

Gets the value of featuresCol or its default value.

`getK`() [source]

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`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.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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).')*

params

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

New in version 1.3.0.

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.')*

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

setK(*value*) [source]

Sets the value of **k**.

New in version 2.0.0.

setMaxIter(*value*)

Sets the value of **maxIter**.

setParams(*self*, *featuresCol="features"*, *predictionCol="prediction"*, *k=2*, *probabilityCol="probability"*, *tol=0.01*, *maxIter=100*, *seed=None*) [source]

Sets params for GaussianMixture.

New in version 2.0.0.

setPredictionCol(*value*)

Sets the value of `predictionCol`.

setProbabilityCol(*value*)

Sets the value of `probabilityCol`.

setSeed(*value*)

Sets the value of `seed`.

setTol(*value*)

Sets the value of `tol`.

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]

Note: Experimental

Model fitted by GaussianMixture.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

gaussiansDF [source]

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.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`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.

weights [source]

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.**LDA**(**args*, ***kwargs*) [source]

Note: Experimental

Latent Dirichlet Allocation (LDA), a topic model designed for text documents.

Terminology:

- "term" = "word": an el
- "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")
>>> model = lda.fit(df)
>>> 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.')*

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getCheckpointInterval`()

Gets the value of checkpointInterval or its default value.

`getDocConcentration`() [source]

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`() [source]

Gets the value of **`k`** or its default value.

New in version 2.0.0.

`getKeepLastCheckpoint`() [source]

Gets the value of **`keepLastCheckpoint`** or its default value.

New in version 2.0.0.

`getLearningDecay`() [source]

Gets the value of **`learningDecay`** or its default value.

New in version 2.0.0.

`getLearningOffset`() [source]

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`() [source]

Gets the value of **`optimizeDocConcentration`** or its default value.

New in version 2.0.0.

getOptimizer() [source]

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getSeed()

Gets the value of seed or its default value.

getSubsamplingRate() [source]

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

New in version 2.0.0.

getTopicConcentration() [source]

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

New in version 2.0.0.

getTopicDistributionCol() [source]

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.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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')*

params

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

New in version 1.3.0.

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)*.

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

setCheckpointInterval(*value*)

Sets the value of **checkpointInterval**.

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*)

Sets the value of **featuresCol**.

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*)

Sets the value of **`maxIter`**.

`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`**. Currenlty only support ‘em’ and ‘online’.

```
>>> algo = LDA().setOptimizer("em")
>>> algo.getOptimizer()
'em'
```

New in version 2.0.0.

`setParams`(**args*, ***kwargs*) [source]

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.

New in version 2.0.0.

`setSeed`(*value*)

Sets the value of **`seed`**.

`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]

> **Note:** Experimental

Latent Dirichlet Allocation (LDA) model. This abstraction permits for different underlying representations, including local and distributed data structures.

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 model with extra params. So both the Python wrapper and the Java model 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.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`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 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.

`params`

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

New in version 1.3.0.

`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]

Note: Experimental

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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

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 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.

`params`

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

New in version 1.3.0.

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)*.

`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]

> **Note:** Experimental

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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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 that 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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

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 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.

`params`

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

New in version 1.3.0.

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)*.

`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.

`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.

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])
```

`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 an numpy.ndarray

values [source]

Returns a list of values

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 array.

`values` = *None*

A list of values corresponding to active entries.

class `pyspark.ml.linalg.`**`Vectors`** [source]

Factory methods for working with vectors. Note that 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 an 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 an 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**(*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"*) [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.

```
>>> 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, maxIter=5, seed=0)
>>> model = als.fit(df)
>>> 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, prediction=-0.13807615637779236)
>>> predictions[1]
Row(user=1, item=0, prediction=2.6258413791656494)
>>> predictions[2]
Row(user=2, item=0, prediction=-1.5018409490585327)
>>> 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.')*

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getAlpha`() [source]

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.

`getFinalStorageLevel`() [source]

Gets the value of finalStorageLevel or its default value.

New in version 2.0.0.

`getImplicitPrefs`() [source]

Gets the value of implicitPrefs or its default value.

New in version 1.4.0.

`getIntermediateStorageLevel`() [source]

Gets the value of intermediateStorageLevel or its default value.

New in version 2.0.0.

`getItemCol`() [source]

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() [source]

Gets the value of nonnegative or its default value.

New in version 1.4.0.

getNumItemBlocks() [source]

Gets the value of numItemBlocks or its default value.

New in version 1.4.0.

getNumUserBlocks() [source]

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getPredictionCol()

Gets the value of predictionCol or its default value.

getRank() [source]

Gets the value of rank or its default value.

New in version 1.4.0.

getRatingCol() [source]

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() [source]

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.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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')*

params

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

New in version 1.3.0.

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.')*

setAlpha(*value*) [source]

Sets the value of **alpha**.

New in version 1.4.0.

setCheckpointInterval(*value*)

Sets the value of **checkpointInterval**.

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*)

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"*) [source]

Sets params for ALS.

New in version 1.4.0.

setPredictionCol(*value*)

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*)

Sets the value of `regParam`.

`setSeed`(*value*)

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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`itemFactors` [source]

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)*.

`params`

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

New in version 1.3.0.

`rank` [source]

rank of the matrix factorization model

New in version 1.4.0.

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)*.

`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.

`userFactors` [source]

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`**(*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*) [source]

Note: Experimental

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),
...     (0.0, Vectors.sparse(1, [], []), 0.0)], ["label", "features", "censor"])
>>> aftsr = AFTSurvivalRegression()
>>> model = aftsr.fit(df)
>>> 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|
|  0.0|(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.

`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.')*

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.')*

`getCensorCol`() [source]

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getPredictionCol()

Gets the value of predictionCol or its default value.

getQuantileProbabilities() [source]

Gets the value of quantileProbabilities or its default value.

New in version 1.6.0.

getQuantilesCol() [source]

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.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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).')*

`params`

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

New in version 1.3.0.

`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)*.

setCensorCol(*value*) [source]

Sets the value of **censorCol**.

New in version 1.6.0.

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

setFitIntercept(*value*)

Sets the value of **fitIntercept**.

setLabelCol(*value*)

Sets the value of **labelCol**.

setMaxIter(*value*)

Sets the value of **maxIter**.

setParams(**args*, ***kwargs*) [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):

New in version 1.6.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

`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*)

Sets the value of **`tol`**.

`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]

Note: Experimental

Model fitted by **`AFTSurvivalRegression`**.

New in version 1.6.0.

`coefficients` [source]

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`intercept` [source]

Model intercept.

New in version 1.6.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

`predict`(*features*) [source]

Predicted value

New in version 2.0.0.

`predictQuantiles`(*features*) [source]

Predicted Quantiles

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)*.

scale [source]

Model scale paramter.

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.regression.**DecisionTreeRegressor**(*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*) [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, varianceCol="variance")
>>> model = dt.fit(df)
>>> model.depth
```

```
1
>>> model.numNodes
3
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
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
```

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.')*

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

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.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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)*.

`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.')*

`params`

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

New in version 1.3.0.

`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.')*

`setCacheNodeIds`(*value*)

Sets the value of **`cacheNodeIds`**.

setCheckpointInterval(*value*)

Sets the value of **`checkpointInterval`**.

setFeaturesCol(*value*)

Sets the value of **`featuresCol`**.

setImpurity(*value*)

Sets the value of **`impurity`**.

New in version 1.4.0.

setLabelCol(*value*)

Sets the value of **`labelCol`**.

setMaxBins(*value*)

Sets the value of **`maxBins`**.

setMaxDepth(*value*)

Sets the value of **`maxDepth`**.

setMaxMemoryInMB(*value*)

Sets the value of **`maxMemoryInMB`**.

setMinInfoGain(*value*)

Sets the value of **`minInfoGain`**.

setMinInstancesPerNode(*value*)

Sets the value of **`minInstancesPerNode`**.

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*) [source]

Sets params for the DecisionTreeRegressor.

New in version 1.4.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`setVarianceCol`(*value*)

Sets the value of **`varianceCol`**.

`supportedImpurities` = *['variance']*

`varianceCol` = *Param(parent='undefined', name='varianceCol', doc='column name for the biased sample variance of prediction.')*

`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.

`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 model with extra params. So both the Python wrapper and the Java model get copied.

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

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`featureImportances` [source]

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.

`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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **load**(*path*)

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

numNodes

Return number of nodes of the decision tree.

New in version 1.5.0.

params

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

New in version 1.3.0.

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)*.

`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.

`write`()

Returns an MLWriter instance for this ML instance.

class `pyspark.ml.regression.`**`GBTRegressor`**(*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"*) [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(maxIter=5, maxDepth=2, seed=42)
>>> print(gbt.getImpurity())
```

```
variance
>>> model = gbt.fit(df)
>>> 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.transform(test0).head().prediction
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 (uid=...) of depth..., DecisionTreeRegressionModel...]
```

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.')*

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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`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.

getLossType() [source]

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.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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)*.

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.')*

`params`

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

New in version 1.3.0.

`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.')*

`setCacheNodeIds`(*value*)

Sets the value of **`cacheNodeIds`**.

`setCheckpointInterval`(*value*)

Sets the value of **`checkpointInterval`**.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setImpurity`(*value*)

Sets the value of **`impurity`**.

New in version 1.4.0.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setLossType`(*value*) [source]

Sets the value of **lossType**.

New in version 1.4.0.

setMaxBins(*value*)

Sets the value of **maxBins**.

setMaxDepth(*value*)

Sets the value of **maxDepth**.

setMaxIter(*value*)

Sets the value of **maxIter**.

setMaxMemoryInMB(*value*)

Sets the value of **maxMemoryInMB**.

setMinInfoGain(*value*)

Sets the value of **minInfoGain**.

setMinInstancesPerNode(*value*)

Sets the value of **minInstancesPerNode**.

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"*) [source]

Sets params for Gradient Boosted Tree Regression.

New in version 1.4.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

setSeed(*value*)

Sets the value of **seed**.

setStepSize(*value*)

Sets the value of **stepSize**.

setSubsamplingRate(*value*)

Sets the value of **subsamplingRate**.

New in version 1.4.0.

stepSize = *Param(parent='undefined', name='stepSize', doc='Step size to be used for each iteration of optimization (>= 0).')*

subsamplingRate = *Param(parent='undefined', name='subsamplingRate', doc='Fraction of the training data used for learning each decision tree, in range (0, 1].')*

supportedImpurities = *['variance']*

supportedLossTypes = *['squared', 'absolute']*

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.

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`featureImportances` [source]

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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`toDebugString`

Full description of model.

New in version 2.0.0.

`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.

`treeWeights`

Return the weights for each tree

New in version 1.5.0.

`trees` [source]

Trees in this ensemble. Warning: These have null parent Estimators.

New in version 2.0.0.

`write`()

Returns an MLWriter instance for this ML instance.

class `pyspark.ml.regression.`**`GeneralizedLinearRegression`**(*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*) [source]

Note: Experimental

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” and “gamma” 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”

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")
>>> model = glr.fit(df)
>>> 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...])
>>> 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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

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 and gamma.')

`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.')*

`getFamily`() [source]

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`() [source]

Gets the value of link or its default value.

New in version 2.0.0.

`getLinkPredictionCol`() [source]

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.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

getWeightCol()

Gets the value of weightCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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.')*

`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).')*

`params`

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

New in version 1.3.0.

`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)*.

`setFamily`(*value*) [source]

Sets the value of **`family`**.

New in version 2.0.0.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setFitIntercept`(*value*)

Sets the value of **`fitIntercept`**.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setLink`(*value*) [source]

Sets the value of **`link`**.

New in version 2.0.0.

`setLinkPredictionCol`(*value*) [source]

Sets the value of **`linkPredictionCol`**.

New in version 2.0.0.

`setMaxIter`(*value*)

Sets the value of **`maxIter`**.

`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*) [source]

Sets params for generalized linear regression.

New in version 2.0.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

setRegParam(*value*)

Sets the value of **regParam**.

setSolver(*value*)

Sets the value of **solver**.

setTol(*value*)

Sets the value of **tol**.

setWeightCol(*value*)

Sets the value of **weightCol**.

solver = *Param(parent='undefined', name='solver', doc="the solver algorithm for optimization. If this is not set or empty, default value is 'auto'.")*

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.**GeneralizedLinearRegressionModel**(*java_model=None*) [source]

Note: Experimental

Model fitted by **GeneralizedLinearRegression**.

New in version 2.0.0.

`coefficients` [source]

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`hasSummary` [source]

Indicates whether a training summary exists for this model instance.

New in version 2.0.0.

`intercept` [source]

Model intercept.

New in version 2.0.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`summary` [source]

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.

`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.`**`GeneralizedLinearRegressionSummary`**(*java_obj=None*) [source]

Note: Experimental

Generalized linear regression results evaluated on a dataset.

New in version 2.0.0.

`aic` [source]

Akaike’s “An Information Criterion”(AIC) for the fitted model.

New in version 2.0.0.

`degreesOfFreedom` [source]

Degrees of freedom.

New in version 2.0.0.

`deviance` [source]

The deviance for the fitted model.

New in version 2.0.0.

`dispersion` [source]

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.

`nullDeviance` [source]

The deviance for the null model.

New in version 2.0.0.

`predictionCol` [source]

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.

`predictions` [source]

Predictions output by the model's *transform* method.

New in version 2.0.0.

`rank` [source]

The numeric rank of the fitted linear model.

New in version 2.0.0.

`residualDegreeOfFreedom` [source]

The residual degrees of freedom.

New in version 2.0.0.

`residualDegreeOfFreedomNull` [source]

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]

Note: Experimental

Generalized linear regression training results.

New in version 2.0.0.

`aic`

Akaike's "An Information Criterion"(AIC) for the fitted model.

New in version 2.0.0.

`coefficientStandardErrors` [source]

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.

`degreesOfFreedom`

Degrees of freedom.

New in version 2.0.0.

`deviance`

The deviance for the fitted model.

New in version 2.0.0.

`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.

`nullDeviance`

The deviance for the null model.

New in version 2.0.0.

`numIterations` [source]

Number of training iterations.

New in version 2.0.0.

`pValues` [source]

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.

`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.

`predictions`

Predictions output by the model's *transform* method.

New in version 2.0.0.

`rank`

The numeric rank of the fitted linear model.

New in version 2.0.0.

`residualDegreeOfFreedom`

The residual degrees of freedom.

New in version 2.0.0.

`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.

solver [source]

The numeric solver used for training.

New in version 2.0.0.

tValues [source]

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**(**args*, ***kwargs*) [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)
>>> 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.

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

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

`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.

`getFeatureIndex`() [source]

Gets the value of featureIndex or its default value.

`getFeaturesCol`()

Gets the value of featuresCol or its default value.

`getIsotonic`() [source]

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

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)*.

params

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

New in version 1.3.0.

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)*.

setFeatureIndex(*value*) [source]

Sets the value of **`featureIndex`**.

setFeaturesCol(*value*)

Sets the value of **`featuresCol`**.

setIsotonic(*value*) [source]

Sets the value of **`isotonic`**.

setLabelCol(*value*)

Sets the value of **`labelCol`**.

setParams(**args*, ***kwargs*) [source]

setParams(self, featuresCol=”features”, labelCol=”label”, predictionCol=”prediction”, weightCol=None, isotonic=True, featureIndex=0): Set the params for IsotonicRegression.

setPredictionCol(*value*)

Sets the value of **`predictionCol`**.

setWeightCol(*value*)

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.regression.`**`IsotonicRegressionModel`**(*java_model=None*) [source]

Model fitted by **`IsotonicRegression`**.

New in version 1.6.0.

`boundaries` [source]

Boundaries in increasing order for which predictions are known.

New in version 1.6.0.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

`predictions` [source]

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)*.

`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.`**`LinearRegression`**(*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*) [source]

Linear regression.

The learning objective is to minimize the squared error, with regularization. The specific squared error loss function used is: L = 1/2n ||A coefficients - y||^2^

This supports multiple types of regularization:

- none (a.k.a. ordinary least squares)
- L2 (ridge regression)
- L1 (Lasso)
- L2 + L1 (elastic net)

```
>>> 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(maxIter=5, regParam=0.0, solver="normal", weightCol="weight")
>>> model = lr.fit(df)
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> abs(model.transform(test0).head().prediction - (-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().prediction - 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
```

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.')*

`getElasticNetParam`()

Gets the value of elasticNetParam 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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`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.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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).')*

`params`

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

New in version 1.3.0.

`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)*.

`setElasticNetParam`(*value*)

Sets the value of **`elasticNetParam`**.

`setFeaturesCol`(*value*)

Sets the value of **`featuresCol`**.

`setFitIntercept`(*value*)

Sets the value of **`fitIntercept`**.

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

setMaxIter(*value*)

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*) [source]

Sets params for linear regression.

New in version 1.4.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

setRegParam(*value*)

Sets the value of **regParam**.

setSolver(*value*)

Sets the value of **solver**.

setStandardization(*value*)

Sets the value of **standardization**.

setTol(*value*)

Sets the value of **tol**.

setWeightCol(*value*)

Sets the value of **weightCol**.

solver = *Param(parent='undefined', name='solver', doc="the solver algorithm for optimization. If this is not set or empty, default value is 'auto'.")*

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.

`coefficients` [source]

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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`hasSummary` [source]

Indicates whether a training summary exists for this model instance.

New in version 2.0.0.

`intercept` [source]

Model intercept.

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`summary` [source]

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.

`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.`**`LinearRegressionSummary`**(*java_obj=None*) [source]

Note: Experimental

Linear regression results evaluated on a dataset.

New in version 2.0.0.

`coefficientStandardErrors` [source]

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.

`devianceResiduals` [source]

The weighted residuals, the usual residuals rescaled by the square root of the instance weights.

New in version 2.0.0.

`explainedVariance` [source]

Returns the explained variance regression score. explainedVariance = 1 - 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.

`featuresCol` [source]

Field in "predictions" which gives the features of each instance as a vector.

New in version 2.0.0.

`labelCol` [source]

Field in "predictions" which gives the true label of each instance.

New in version 2.0.0.

`meanAbsoluteError` [source]

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.

`meanSquaredError` [source]

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.

`numInstances` [source]

Number of instances in DataFrame predictions

New in version 2.0.0.

`pValues` [source]

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.

`predictionCol` [source]

Field in “predictions” which gives the predicted value of the label at each instance.

New in version 2.0.0.

`predictions` [source]

Dataframe outputted by the model’s *transform* method.

New in version 2.0.0.

`r2` [source]

Returns R^2^, the coefficient of determination.

See also: *Wikipedia coefficient of determination <http://en.wikipedia.org/wiki/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.

`residuals` [source]

Residuals (label - predicted value)

New in version 2.0.0.

`rootMeanSquaredError` [source]

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.

`tValues` [source]

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]

Note: Experimental

Linear regression training results. Currently, the training summary ignores the training weights except for the objective trace.

New in version 2.0.0.

`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.

`devianceResiduals`

The weighted residuals, the usual residuals rescaled by the square root of the instance weights.

New in version 2.0.0.

`explainedVariance`

Returns the explained variance regression score. explainedVariance = 1 - 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.

`featuresCol`

Field in “predictions” which gives the features of each instance as a vector.

New in version 2.0.0.

`labelCol`

Field in “predictions” which gives the true label of each instance.

New in version 2.0.0.

`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.

`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.

`numInstances`

Number of instances in DataFrame predictions

New in version 2.0.0.

`objectiveHistory` [source]

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.

`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.

`predictionCol`

Field in "predictions" which gives the predicted value of the label at each instance.

New in version 2.0.0.

`predictions`

Dataframe outputted by the model's *transform* method.

New in version 2.0.0.

`r2`

Returns R^2^, the coefficient of determination.

See also: *Wikipedia coefficient of determination <http://en.wikipedia.org/wiki/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.

`residuals`

Residuals (label - predicted value)

New in version 2.0.0.

`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.

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.

totalIterations [source]

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**(*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"*) [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, seed=42)
>>> model = rf.fit(df)
>>> model.featureImportances
SparseVector(1, {0: 1.0})
>>> allclose(model.treeWeights, [1.0, 1.0])
True
>>> test0 = spark.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> model.trees
[DecisionTreeRegressionModel (uid=...) of 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.')*

`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

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`featureSubsetStrategy` = *Param(parent='undefined', name='featureSubsetStrategy', doc='The number of features to consider for splits at each tree node. Supported options: auto, all, onethird, sqrt, log2 (0.0-1.0], [1-n].')*

`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.

`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.

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.

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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

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.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`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)*.

`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.')*

`numTrees` = *Param(parent='undefined', name='numTrees', doc='Number of trees to train (>= 1).')*

`params`

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

New in version 1.3.0.

`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.')*

setCacheNodeIds(*value*)

Sets the value of `cacheNodeIds`.

setCheckpointInterval(*value*)

Sets the value of `checkpointInterval`.

setFeatureSubsetStrategy(*value*)

Sets the value of `featureSubsetStrategy`.

New in version 1.4.0.

setFeaturesCol(*value*)

Sets the value of `featuresCol`.

setImpurity(*value*)

Sets the value of `impurity`.

New in version 1.4.0.

setLabelCol(*value*)

Sets the value of `labelCol`.

setMaxBins(*value*)

Sets the value of `maxBins`.

setMaxDepth(*value*)

Sets the value of `maxDepth`.

setMaxMemoryInMB(*value*)

Sets the value of `maxMemoryInMB`.

`setMinInfoGain`(*value*)

Sets the value of **`minInfoGain`**.

`setMinInstancesPerNode`(*value*)

Sets the value of **`minInstancesPerNode`**.

`setNumTrees`(*value*)

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"*) [source]

Sets params for linear regression.

New in version 1.4.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`setSubsamplingRate`(*value*)

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']* `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.

`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 model with extra params. So both the Python wrapper and the Java model 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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`featureImportances` [source]

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.

`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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

classmethod **`load`**(*path*)

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

`params`

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

New in version 1.3.0.

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)*.

`toDebugString`

Full description of model.

New in version 2.0.0.

`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.

`treeWeights`

Return the weights for each tree

New in version 1.5.0.

`trees` [source]

Trees in this ensemble. Warning: These have null parent Estimators.

New in version 2.0.0.

`write`()

Returns an MLWriter instance for this ML instance.

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.

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**(*self*, *estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *numFolds=3*, *seed=None*) [source]

K-fold cross validation.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> from pyspark.ml.linalg import Vectors
>>> 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)
>>> cvModel = cv.fit(dataset)
>>> evaluator.evaluate(cvModel.transform(dataset))
0.8333...
```

New in version 1.4.0.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`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.

`getEstimator`()

Gets the value of estimator or its default value.

`getEstimatorParamMaps`()

Gets the value of estimatorParamMaps or its default value.

`getEvaluator`()

Gets the value of evaluator or its default value.

`getNumFolds`() [source]

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getSeed`()

Gets the value of seed or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

numFolds = *Param(parent='undefined', name='numFolds', doc='number of folds for cross validation')*

params

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

New in version 1.3.0.

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

setEstimator(*value*)

Sets the value of `estimator`.

setEstimatorParamMaps(*value*)

Sets the value of `estimatorParamMaps`.

setEvaluator(*value*)

Sets the value of **`evaluator`**.

`setNumFolds`(*value*) [source]

Sets the value of **`numFolds`**.

New in version 1.4.0.

`setParams`(**args*, ***kwargs*) [source]

setParams(self, estimator=None, estimatorParamMaps=None, evaluator=None, numFolds=3, seed=None): Sets params for cross validator.

New in version 1.4.0.

`setSeed`(*value*)

Sets the value of **`seed`**.

class `pyspark.ml.tuning.`**`CrossValidatorModel`**(*bestModel*, *avgMetrics=[]*) [source]

Model from k-fold cross validation.

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

`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.

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`getEstimator`()

Gets the value of estimator or its default value.

`getEstimatorParamMaps`()

Gets the value of estimatorParamMaps or its default value.

`getEvaluator`()

Gets the value of evaluator 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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getSeed()

Gets the value of seed or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

params

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

New in version 1.3.0.

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

setEstimator(*value*)

Sets the value of **`estimator`**.

`setEstimatorParamMaps`(*value*)

Sets the value of **`estimatorParamMaps`**.

`setEvaluator`(*value*)

Sets the value of **`evaluator`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`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.tuning.`**`TrainValidationSplit`**(*self*, *estimator=None*, *estimatorParamMaps=None*, *evaluator=None*, *trainRatio=0.75*, *seed=None*) [source]

Note: Experimental

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
>>> 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()
>>> tvs = TrainValidationSplit(estimator=lr, estimatorParamMaps=grid, evaluator=evaluator)
>>> tvsModel = tvs.fit(dataset)
>>> evaluator.evaluate(tvsModel.transform(dataset))
0.8333...
```

New in version 2.0.0.

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.

New in version 1.4.0.

explainParams()

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

New in version 1.4.0.

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

New in version 1.4.0.

`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.

`getEstimator`()

Gets the value of estimator or its default value.

`getEstimatorParamMaps`()

Gets the value of estimatorParamMaps or its default value.

`getEvaluator`()

Gets the value of evaluator 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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getSeed`()

Gets the value of seed or its default value.

`getTrainRatio`() [source]

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.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`isSet`(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`params`

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

New in version 1.3.0.

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

`setEstimator`(*value*)

Sets the value of `estimator`.

`setEstimatorParamMaps`(*value*)

Sets the value of **`estimatorParamMaps`**.

`setEvaluator`(*value*)

Sets the value of **`evaluator`**.

`setParams`(**args*, ***kwargs*) [source]

setParams(self, estimator=None, estimatorParamMaps=None, evaluator=None, trainRatio=0.75, seed=None): Sets params for the train validation split.

New in version 2.0.0.

`setSeed`(*value*)

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.')*

class `pyspark.ml.tuning.`**`TrainValidationSplitModel`**(*bestModel*, *validationMetrics=[]*) [source]

Note: Experimental

Model from train validation split.

New in version 2.0.0.

`bestModel` = *None*

best model from cross validation

`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.

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`getEstimator`()

Gets the value of estimator or its default value.

`getEstimatorParamMaps`()

Gets the value of estimatorParamMaps or its default value.

getEvaluator()

Gets the value of evaluator 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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getSeed()

Gets the value of seed or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

isSet(*param*)

Checks whether a param is explicitly set by user.

New in version 1.4.0.

`params`

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

New in version 1.3.0.

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

`setEstimator`(*value*)

Sets the value of **`estimator`**.

`setEstimatorParamMaps`(*value*)

Sets the value of **`estimatorParamMaps`**.

`setEvaluator`(*value*)

Sets the value of **`evaluator`**.

`setSeed`(*value*)

Sets the value of **`seed`**.

`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

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.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

params

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

New in version 1.3.0.

class pyspark.ml.evaluation.**BinaryClassificationEvaluator**(*self*, *rawPredictionCol="rawPrediction"*, *labelCol="label"*, *metricName="areaUnderROC"*) [source]

Note: Experimental

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(rawPredictionCol="raw")
>>> evaluator.evaluate(dataset)
0.70...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "areaUnderPR"})
0.83...
```

New in version 1.4.0.

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

New in version 1.4.0.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`getLabelCol`()

Gets the value of labelCol or its default value.

`getMetricName`() [source]

Gets the value of metricName or its default value.

New in version 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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

`getRawPredictionCol`()

Gets the value of rawPredictionCol or its default value.

`hasDefault`(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

`hasParam`(*paramName*)

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

New in version 1.4.0.

`isDefined`(*param*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

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

`metricName` = *Param(parent='undefined', name='metricName', doc='metric name in evaluation (areaUnderROC|areaUnderPR)')*

`params`

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

New in version 1.3.0.

`rawPredictionCol` = *Param(parent='undefined', name='rawPredictionCol', doc='raw prediction (a.k.a. confidence) column name.')*

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setMetricName`(*value*) [source]

Sets the value of **`metricName`**.

New in version 1.4.0.

`setParams`(*self*, *rawPredictionCol="rawPrediction"*, *labelCol="label"*, *metricName="areaUnderROC"*) [source]

Sets params for binary classification evaluator.

New in version 1.4.0.

`setRawPredictionCol`(*value*)

Sets the value of **`rawPredictionCol`**.

class `pyspark.ml.evaluation.`**`RegressionEvaluator`**(*self*, *predictionCol="prediction"*, *labelCol="label"*, *metricName="rmse"*) [source]

Note: Experimental

Evaluator for Regression, which expects two input columns: prediction and label.

```
>>> 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(predictionCol="raw")
>>> evaluator.evaluate(dataset)
2.842...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "r2"})
0.993...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "mae"})
2.649...
```

New in version 1.4.0.

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

New in version 1.4.0.

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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

`getLabelCol`()

Gets the value of labelCol or its default value.

`getMetricName`() [source]

Gets the value of metricName or its default value.

New in version 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.

New in version 1.4.0.

`getParam`(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getPredictionCol()

Gets the value of predictionCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

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.

New in version 1.4.0.

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

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.')*

params

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

New in version 1.3.0.

`predictionCol` = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

`setLabelCol`(*value*)

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"*) [source]

Sets params for regression evaluator.

New in version 1.4.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.

class `pyspark.ml.evaluation.`**`MulticlassClassificationEvaluator`**(*self*, *predictionCol="prediction"*, *labelCol="label"*, *metricName="f1"*) [source]

Note: Experimental

Evaluator for Multiclass Classification, which expects two input columns: prediction and label.

```
>>> 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(predictionCol="prediction")
>>> evaluator.evaluate(dataset)
0.66...
>>> evaluator.evaluate(dataset, {evaluator.metricName: "accuracy"})
0.66...
```

New in version 1.5.0.

`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

New in version 1.4.0.

`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.

New in version 1.4.0.

`explainParams`()

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

New in version 1.4.0.

`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

New in version 1.4.0.

getLabelCol()

Gets the value of labelCol or its default value.

getMetricName() [source]

Gets the value of metricName or its default value.

New in version 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.

New in version 1.4.0.

getParam(*paramName*)

Gets a param by its name.

New in version 1.4.0.

getPredictionCol()

Gets the value of predictionCol or its default value.

hasDefault(*param*)

Checks whether a param has a default value.

New in version 1.4.0.

hasParam(*paramName*)

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

New in version 1.4.0.

isDefined(*param*)

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

New in version 1.4.0.

`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.

New in version 1.4.0.

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

`metricName` = *Param(parent='undefined', name='metricName', doc='metric name in evaluation (f1|weightedPrecision|weightedRecall|accuracy)')*

`params`

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

New in version 1.3.0.

`predictionCol` = *Param(parent='undefined', name='predictionCol', doc='prediction column name.')*

`setLabelCol`(*value*)

Sets the value of **`labelCol`**.

`setMetricName`(*value*) [source]

Sets the value of **`metricName`**.

New in version 1.5.0.

`setParams`(*self*, *predictionCol="prediction"*, *labelCol="label"*, *metricName="f1"*) [source]

Sets params for multiclass classification evaluator.

New in version 1.5.0.

`setPredictionCol`(*value*)

Sets the value of **`predictionCol`**.