pyspark.ml package

ML Pipeline APIs

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 an **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.

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.

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.

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.

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.

pyspark.ml.param module

class pyspark.ml.param.**Param**(*parent*, *name*, *doc*) [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.

pyspark.ml.feature module

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

> **Note:** Experimental

Binarize a column of continuous features given a threshold.

```
>>> df = sqlContext.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
```

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.

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.

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.

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

> **Note:** Experimental

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

```
>>> df = sqlContext.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
```

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.

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.

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.

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 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.')*

param for Splitting points for mapping continuous features into buckets. With n+1 splits,

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.feature.**CountVectorizer**(*self*, *minTF=1.0*, *minDF=1.0*, *vocabSize=1 << 18*, *inputCol=None*, *outputCol=None*) [source]

> **Note:** Experimental

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

```
>>> df = sqlContext.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(map(str, model.vocabulary))
['a', 'b', 'c']
```

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.

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.

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.

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*, *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.')*

class pyspark.ml.feature.**CountVectorizerModel**(*java_model*) [source]

> **Note:** Experimental

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.

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.

vocabulary [source]

An array of terms in the vocabulary.

New in version 1.6.0.

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

> **Note:** Experimental

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).

More information on *https://en.wikipedia.org/wiki/Discrete_cosine_transform#DCT-II Wikipedia*.

```
>>> from pyspark.mllib.linalg import Vectors
>>> df1 = sqlContext.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])
```

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.

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.

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.

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

Note: Experimental

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.mllib.linalg import Vectors
>>> df = sqlContext.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])
```

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 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.

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.

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.

scalingVec = *Param(parent='undefined', name='scalingVec', doc='vector for hadamard product, it must be MLlib Vector type.')*

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 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.

class pyspark.ml.feature.**HashingTF**(*self*, *numFeatures=1 << 18*, *inputCol=None*, *outputCol=None*) [source]

Note: Experimental

Maps a sequence of terms to their term frequencies using the hashing trick.

```
>>> df = sqlContext.createDataFrame([(["a", "b", "c"],)], ["words"])
>>> hashingTF = HashingTF(numFeatures=10, inputCol="words", outputCol="features")
>>> hashingTF.transform(df).head().features
SparseVector(10, {7: 1.0, 8: 1.0, 9: 1.0})
>>> hashingTF.setParams(outputCol="freqs").transform(df).head().freqs
SparseVector(10, {7: 1.0, 8: 1.0, 9: 1.0})
>>> params = {hashingTF.numFeatures: 5, hashingTF.outputCol: "vector"}
>>> hashingTF.transform(df, params).head().vector
SparseVector(5, {2: 1.0, 3: 1.0, 4: 1.0})
```

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.

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.

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.

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*, *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.

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

> **Note:** Experimental

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

```
>>> from pyspark.mllib.linalg import DenseVector
>>> df = sqlContext.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")
>>> idf.fit(df).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])
```

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.

minDocFreq = *Param(parent='undefined', name='minDocFreq', doc='minimum 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.

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.

class pyspark.ml.feature.**IDFModel**(*java_model*) [source]

> **Note:** Experimental

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.

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.feature.**IndexToString**(*self*, *inputCol=None*, *outputCol=None*, *labels=None*) [source]

Note: Experimental

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.')*

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.

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.

class pyspark.ml.feature.**MinMaxScaler**(*self*, *min=0.0*, *max=1.0*, *inputCol=None*, *outputCol=None*) [source]

Note: Experimental

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.mllib.linalg import Vectors
>>> df = sqlContext.createDataFrame([(Vectors.dense([0.0]),), (Vectors.dense([2.0]),)], ["a"])
>>> mmScaler = MinMaxScaler(inputCol="a", outputCol="scaled")
>>> model = mmScaler.fit(df)
>>> model.transform(df).show()
+-----+------+
|    a|scaled|
+-----+------+
|[0.0]| [0.0]|
|[2.0]| [1.0]|
+-----+------+
...
```

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.

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.

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.

class pyspark.ml.feature.**MinMaxScalerModel**(*java_model*) [source]

> **Note:** Experimental

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.

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.feature.**NGram**(*self*, *n=2*, *inputCol=None*, *outputCol=None*) [source]

> **Note:** Experimental

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 = sqlContext.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.
```

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.

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.

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.

class pyspark.ml.feature.**Normalizer**(*self*, *p=2.0*, *inputCol=None*, *outputCol=None*) [source]

Note: Experimental

Normalize a vector to have unit norm using the given p-norm.

```
>>> from pyspark.mllib.linalg import Vectors
>>> svec = Vectors.sparse(4, {1: 4.0, 3: 3.0})
>>> df = sqlContext.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])
```

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.

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.

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.

class pyspark.ml.feature.**OneHotEncoder**(*self*, *includeFirst=True*, *inputCol=None*, *outputCol=None*) [source]

Note: Experimental

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})
```

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.

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.

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.

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

> **Note:** Experimental

PCA trains a model to project vectors to a low-dimensional space using PCA.

```
>>> from pyspark.mllib.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 = sqlContext.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...])
```

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')*

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.

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.

class pyspark.ml.feature.**PCAModel**(*java_model*) [source]

Note: Experimental

Model fitted by PCA.

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.

`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.feature.**PolynomialExpansion**(*self*, *degree=2*, *inputCol=None*, *outputCol=None*) [source]

> **Note:** Experimental

Perform feature expansion in a polynomial space. As said in wikipedia of Polynomial Expansion, which is available at *http://en.wikipedia.org/wiki/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.mllib.linalg import Vectors
>>> df = sqlContext.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])
```

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.

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.

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.

class pyspark.ml.feature.**RegexTokenizer**(*self*, *minTokenLength=1*, *gaps=True*, *pattern="s+"*, *inputCol=None*, *outputCol=None*) [source]

> **Note:** Experimental

A regex based tokenizer that extracts tokens either by using the provided regex pattern (in Java dialect) to split the text (default) or repeatedly matching the regex (if gaps is false). Optional parameters also allow filtering tokens using a minimal length. It returns an array of strings that can be empty.

```
>>> df = sqlContext.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.
```

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')*

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.

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.

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')*

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*) [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.

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.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: http://stat.ethz.ch/R-manual/R-patched/library/stats/html/formula.html

```
>>> df = sqlContext.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")
>>> rf.fit(df).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|
+---+---+---+--------+-----+
...
```

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.')*

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.

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.

class pyspark.ml.feature.**RFormulaModel**(*java_model*) [source]

> **Note:** Experimental

Model fitted by **RFormula**.

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.

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.feature.**SQLTransformer**(*self*, *statement=None*) [source]

Note: Experimental

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 = sqlContext.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)
```

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.

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.

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.

class pyspark.ml.feature.**StandardScaler**(*self*, *withMean=False*, *withStd=True*, *inputCol=None*, *outputCol=None*) [source]

> **Note:** Experimental

Standardizes features by removing the mean and scaling to unit variance using column summary statistics on the samples in the training set.

```
>>> from pyspark.mllib.linalg import Vectors
>>> df = sqlContext.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])
```

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.

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.

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')*

class pyspark.ml.feature.**StandardScalerModel**(*java_model*) [source]

> **Note:** Experimental

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.

mean [source]

Mean of the StandardScalerModel.

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.

std [source]

Standard deviation of the StandardScalerModel.

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.

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

> **Note:** Experimental

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

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]

Get whether to do a case sensitive comparison over the stop words.

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]

Get the stopwords.

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.

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.

setCaseSensitive(*value*) [source]

Set whether to do a case sensitive comparison over the stop words

New in version 1.6.0.

setInputCol(*value*)

Sets the value of **inputCol**.

setOutputCol(*value*)

Sets the value of **outputCol**.

setParams(*self*, *inputCol="input"*, *outputCol="output"*, *stopWords=None*, *caseSensitive=false*) [source]

Sets params for this StopWordRemover.

New in version 1.6.0.

setStopWords(*value*) [source]

Specify the stopwords to be filtered.

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.

class pyspark.ml.feature.**StringIndexer**(*self*, *inputCol=None*, *outputCol=None*, *handleInvalid="error"*) [source]

> **Note:** Experimental

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")
>>> 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')]
```

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 errror). 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.

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.

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.

class pyspark.ml.feature.**StringIndexerModel**(*java_model*) [source]

> **Note:** Experimental

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.

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.feature.**Tokenizer**(*self*, *inputCol=None*, *outputCol=None*) [source]

> **Note:** Experimental

A tokenizer that converts the input string to lowercase and then splits it by white spaces.

```
>>> df = sqlContext.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.
```

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.

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.

setInputCol(*value*)

Sets the value of **inputCol**.

setOutputCol(*value*)

Sets the value of **outputCol**.

setParams(*self*, *inputCol="input"*, *outputCol="output"*) [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.

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

Note: Experimental

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

```
>>> df = sqlContext.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])
```

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.

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.

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.

class pyspark.ml.feature.**VectorIndexer**(*self*, *maxCategories=20*, *inputCol=None*, *outputCol=None*) [source]

> **Note:** Experimental

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.mllib.linalg import Vectors
```

```
>>> df = sqlContext.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])
```

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.

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.

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.

class pyspark.ml.feature.**VectorSlicer**(*self*, *inputCol=None*, *outputCol=None*, *indices=None*, *names=None*) [source]

> **Note:** Experimental

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.mllib.linalg import Vectors
>>> df = sqlContext.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])
```

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.

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.

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.

class pyspark.ml.feature.**Word2Vec**(*self*, *vectorSize=100*, *minCount=5*, *numPartitions=1*, *stepSize=0.025*, *maxIter=1*, *seed=None*, *inputCol=None*, *outputCol=None*) [source]

Note: Experimental

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 = sqlContext.createDataFrame([(sent,), (sent,)], ["sentence"])
>>> model = Word2Vec(vectorSize=5, seed=42, inputCol="sentence", outputCol="model").fit(doc)
>>> model.getVectors().show()
+----+--------------------+
|word|              vector|
+----+--------------------+
|   a|[0.09461779892444...|
|   b|[1.15474212169647...|
|   c|[-0.3794820010662...|
+----+--------------------+
...
>>> model.findSynonyms("a", 2).show()
+----+--------------------+
|word|          similarity|
+----+--------------------+
|   b| 0.16782984556103436|
|   c|-0.46761559092107646|
+----+--------------------+
...
>>> model.transform(doc).head().model
DenseVector([0.5524, -0.4995, -0.3599, 0.0241, 0.3461])
```

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.

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.

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.

maxIter = *Param(parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')*

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.

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

setInputCol(*value*)

Sets the value of **inputCol**.

setMaxIter(*value*)

Sets the value of **maxIter**.

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*) [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.

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

vectorSize = *Param(parent='undefined', name='vectorSize', doc='the dimension of codes after transforming from words')*

class pyspark.ml.feature.**Word2VecModel**(*java_model*) [source]

> **Note:** Experimental

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.

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.

pyspark.ml.classification module

class pyspark.ml.classification.**LogisticRegression**(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *maxIter=100*, *regParam=0.1*, *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.mllib.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.
```

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]

Gets the value of threshold or its default value.

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.')*

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.')*

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

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=0.1*, *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.')*

param for threshold in binary classification, in range [0, 1].

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.')*

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.')*

class pyspark.ml.classification.**LogisticRegressionModel**(*java_model*) [source]

Model fitted by LogisticRegression.

New in version 1.3.0.

coefficients [source]

Model coefficients.

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.

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.

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.

weights [source]

Model weights.

New in version 1.4.0.

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"*) [source]

http://en.wikipedia.org/wiki/Decision_tree_learning Decision tree learning algorithm for classification. It supports both binary and multiclass labels, as well as both continuous and categorical features.

```
>>> from pyspark.mllib.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = sqlContext.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
>>> test0 = sqlContext.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 = sqlContext.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
```

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 1.6.0.

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.

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.')*

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.')*

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.')*

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"*) [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**.

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

class pyspark.ml.classification.**DecisionTreeClassificationModel**(*java_model*) [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.

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.

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.

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.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*) [source]

http://en.wikipedia.org/wiki/Gradient_boosting 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.

```
>>> from numpy import allclose
>>> from pyspark.mllib.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = sqlContext.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")
>>> model = gbt.fit(td)
>>> allclose(model.treeWeights, [1.0, 0.1, 0.1, 0.1, 0.1])
True
>>> test0 = sqlContext.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
```

```
>>> model.transform(test0).head().prediction
0.0
>>> test1 = sqlContext.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
```

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.

`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.')*

lossType = *Param(parent='undefined', name='lossType', doc='Loss function which GBT tries to minimize (case-insensitive). Supported options: logistic')*

param for Loss function which GBT tries to minimize (case-insensitive).

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.')*

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.')*

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*) [source]

Sets params for Gradient Boosted Tree Classification.

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.')*

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

supportedLossTypes = *['logistic']*

class pyspark.ml.classification.**GBTClassificationModel**(*java_model*) [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.

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.

treeWeights

Return the weights for each tree

New in version 1.5.0.

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]

http://en.wikipedia.org/wiki/Random_forest 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.mllib.linalg import Vectors
>>> from pyspark.ml.feature import StringIndexer
>>> df = sqlContext.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)
>>> allclose(model.treeWeights, [1.0, 1.0, 1.0])
True
```

```
>>> test0 = sqlContext.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 = sqlContext.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
```

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.

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')*

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.')*

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.')*

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.')*

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']*

class pyspark.ml.classification.**RandomForestClassificationModel**(*java_model*) [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.

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.

treeWeights

Return the weights for each tree

New in version 1.5.0.

class pyspark.ml.classification.**NaiveBayes**(*self*, *featuresCol="features"*, *labelCol="label"*, *predictionCol="prediction"*, *probabilityCol="probability"*, *rawPredictionCol="rawPrediction"*, *smoothing=1.0*, *modelType="multinomial"*) [source]

Naive Bayes Classifiers. It supports both Multinomial and Bernoulli NB. Multinomial NB (*http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html*) 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 (*http://nlp.stanford.edu/IR-book/html/htmledition/the-bernoulli-model-1.html*). The input feature values must be nonnegative.

```
>>> from pyspark.sql import Row
>>> from pyspark.mllib.linalg import Vectors
>>> df = sqlContext.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
```

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.

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.')*

modelType = *Param(parent='undefined', name='modelType', doc='The model type which is a string (case-sensitive). Supported options: multinomial (default) and bernoulli.')*

param for the model type.

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.')*

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"*) [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.

smoothing = *Param(parent='undefined', name='smoothing', doc='The smoothing parameter, should be >= 0, default is 1.0')*

param for the smoothing parameter.

class pyspark.ml.classification.**NaiveBayesModel**(*java_model*) [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.

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 1.5.0.

theta [source]

log of class conditional probabilities.

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.

class pyspark.ml.classification.**MultilayerPerceptronClassifier**(*self, featuresCol="features", labelCol="label", predictionCol="prediction", maxIter=100, tol=1e-4, seed=None, layers=[1, 1], blockSize=128*) [source]

Classifier trainer based on the Multilayer Perceptron. Each layer has sigmoid activation function, output layer has softmax. Number of inputs has to be equal to the size of feature vectors. Number of outputs has to be equal to the total number of labels.

```
>>> from pyspark.mllib.linalg import Vectors
>>> df = sqlContext.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, 5, 2], blockSize=1, seed=11)
>>> model = mlp.fit(df)
>>> model.layers
[2, 5, 2]
```

```
>>> model.weights.size
27
>>> testDF = sqlContext.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|
+---------+----------+
...
```

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.

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.

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.')*

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, default is [1, 1].')*

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.')*

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**.

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=[1, 1], blockSize=128*) [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**.

setTol(*value*)

Sets the value of **tol**.

tol = *Param(parent='undefined', name='tol', doc='the convergence tolerance for iterative algorithms.')*

class pyspark.ml.classification.**MultilayerPerceptronClassificationModel**(*java_model*) [source]

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.

`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.

`weights` [source]

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

New in version 1.6.0.

pyspark.ml.clustering module

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 support for multiple parallel runs and a k-means++ like initialization mode (the k-means|| algorithm by Bahmani et al). When multiple concurrent runs are requested, they are executed together with joint passes over the data for efficiency.

```
>>> from pyspark.mllib.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 = sqlContext.createDataFrame(data, ["features"])
>>> kmeans = KMeans(k=2, seed=1)
>>> model = kmeans.fit(df)
>>> centers = model.clusterCenters()
>>> len(centers)
2
>>> transformed = model.transform(df).select("features", "prediction")
>>> rows = transformed.collect()
>>> rows[0].prediction == rows[1].prediction
True
>>> rows[2].prediction == rows[3].prediction
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.

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='steps for k-means initialization mode')*

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 clusters to create')*

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.')*

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

setFeaturesCol(*value*)

Sets the value of **featuresCol**.

setInitMode(*value*) [source]

Sets the value of **initMode**.

```
>>> algo = KMeans()
>>> algo.getInitMode()
'k-means||'
>>> algo = algo.setInitMode("random")
>>> algo.getInitMode()
'random'
```

New in version 1.5.0.

setInitSteps(*value*) [source]

Sets the value of **initSteps**.

```
>>> algo = KMeans().setInitSteps(10)
>>> algo.getInitSteps()
10
```

New in version 1.5.0.

setK(*value*) [source]

Sets the value of **k**.

```
>>> algo = KMeans().setK(10)
>>> algo.getK()
10
```

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.')*

class pyspark.ml.clustering.**KMeansModel**(*java_model*) [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.

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.

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.

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*) [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”, available at *http://dx.doi.org/10.1109/ICDM.2008.22*, 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 = sqlContext.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)
>>> model = als.fit(df)
>>> model.rank
10
>>> model.userFactors.orderBy("id").collect()
[Row(id=0, features=[...]), Row(id=1, ...), Row(id=2, ...)]
>>> test = sqlContext.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)
```

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.

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.

getImplicitPrefs() [source]

Gets the value of implicitPrefs or its default value.

New in version 1.4.0.

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')*

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')*

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')*

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

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**.

setImplicitPrefs(*value*) [source]

Sets the value of **implicitPrefs**.

New in version 1.4.0.

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*) [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')*

class pyspark.ml.recommendation.**ALSModel**(*java_model*) [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.

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.

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.

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]

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.mllib.linalg import Vectors
>>> df = sqlContext.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|
+-----+---------+------+----------+
...
```

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.')*

Param for censor column name

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.')*

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.')*

Param for quantile probabilities array

quantilesCol = *Param(parent='undefined', name='quantilesCol', doc='quantiles column name. This column will output quantiles of corresponding quantileProbabilities if it is set.')*

Param for quantiles column name

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.')*

class pyspark.ml.regression.**AFTSurvivalRegressionModel**(*java_model*) [source]

Model fitted by AFTSurvivalRegression.

New in version 1.6.0.

coefficients [source]

Model coefficients.

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.

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.

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

predictQuantiles(*features*) [source]

Predicted Quantiles

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.

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"*) [source]

http://en.wikipedia.org/wiki/Decision_tree_learning Decision tree learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from pyspark.mllib.linalg import Vectors
>>> df = sqlContext.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> dt = DecisionTreeRegressor(maxDepth=2)
>>> model = dt.fit(df)
>>> model.depth
```

```
1
>>> model.numNodes
3
>>> test0 = sqlContext.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> test1 = sqlContext.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
```

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.

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.')*

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.')*

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.')*

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"*) [source]

Sets params for the DecisionTreeRegressor.

New in version 1.4.0.

setPredictionCol(*value*)

Sets the value of **predictionCol**.

supportedImpurities = *['variance']*

class pyspark.ml.regression.**DecisionTreeRegressionModel**(*java_model*) [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.

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.

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.

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.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*) [source]

http://en.wikipedia.org/wiki/Gradient_boosting Gradient-Boosted Trees (GBTs) learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from numpy import allclose
>>> from pyspark.mllib.linalg import Vectors
>>> df = sqlContext.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> gbt = GBTRegressor(maxIter=5, maxDepth=2)
>>> model = gbt.fit(df)
>>> allclose(model.treeWeights, [1.0, 0.1, 0.1, 0.1, 0.1])
True
>>> test0 = sqlContext.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
```

```
>>> test1 = sqlContext.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
1.0
```

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.

`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.')*

lossType = *Param(parent='undefined', name='lossType', doc='Loss function which GBT tries to minimize (case-insensitive). Supported options: squared, absolute')*

param for Loss function which GBT tries to minimize (case-insensitive).

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.')*

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.')*

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*, *subsamplingRate=1.0*, *checkpointInterval=10*, *lossType="squared"*, *maxIter=20*, *stepSize=0.1*) [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.')*

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

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

class pyspark.ml.regression.**GBTRegressionModel**(*java_model*) [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.

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.

treeWeights

Return the weights for each tree

New in version 1.5.0.

class pyspark.ml.regression.**IsotonicRegression**(**args*, ***kwargs*) [source]

Note: Experimental

Currently implemented using parallelized pool adjacent violators algorithm. Only univariate (single feature) algorithm supported.

```
>>> from pyspark.mllib.linalg import Vectors
>>> df = sqlContext.createDataFrame([
...     (1.0, Vectors.dense(1.0)),
...     (0.0, Vectors.sparse(1, [], []))], ["label", "features"])
>>> ir = IsotonicRegression()
>>> model = ir.fit(df)
>>> test0 = sqlContext.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> model.boundaries
DenseVector([0.0, 1.0])
```

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.')*

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.')*

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.')*

class pyspark.ml.regression.**IsotonicRegressionModel**(*java_model*) [source]

> **Note:** Experimental

Model fitted by IsotonicRegression.

boundaries [source]

Model boundaries.

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.

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.

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.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 support multiple types of regularization:

- none (a.k.a. ordinary least squares)

- L2 (ridge regression)
- L1 (Lasso)
- L2 + L1 (elastic net)

```
>>> from pyspark.mllib.linalg import Vectors
>>> df = sqlContext.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 = sqlContext.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 = sqlContext.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.
```

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.')*

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.')*

regParam = *Param(parent='undefined', name='regParam', doc='regularization parameter (>= 0).')*

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.')*

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.')*

class pyspark.ml.regression.**LinearRegressionModel**(*java_model*) [source]

Model fitted by LinearRegression.

New in version 1.4.0.

coefficients [source]

Model coefficients.

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.

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.

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.

weights [source]

Model weights.

New in version 1.4.0.

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]

http://en.wikipedia.org/wiki/Random_forest Random Forest learning algorithm for regression. It supports both continuous and categorical features.

```
>>> from numpy import allclose
>>> from pyspark.mllib.linalg import Vectors
>>> df = sqlContext.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)
>>> allclose(model.treeWeights, [1.0, 1.0])
True
>>> test0 = sqlContext.createDataFrame([(Vectors.dense(-1.0),)], ["features"])
>>> model.transform(test0).head().prediction
0.0
>>> test1 = sqlContext.createDataFrame([(Vectors.sparse(1, [0], [1.0]),)], ["features"])
>>> model.transform(test1).head().prediction
0.5
```

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')*

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.')*

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.')*

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.')*

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']*

class pyspark.ml.regression.**RandomForestRegressionModel**(*java_model*) [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.

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.

treeWeights

Return the weights for each tree

New in version 1.5.0.

pyspark.ml.tuning module

class pyspark.ml.tuning.**ParamGridBuilder** [source]

Builder for a param grid used in grid search-based model selection.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> lr = LogisticRegression()
>>> output = ParamGridBuilder() \
...     .baseOn({lr.labelCol: 'l'}) \
...     .baseOn([lr.predictionCol, 'p']) \
...     .addGrid(lr.regParam, [1.0, 2.0]) \
...     .addGrid(lr.maxIter, [1, 5]) \
...     .build()
>>> expected = [
...     {lr.regParam: 1.0, lr.maxIter: 1, lr.labelCol: 'l', lr.predictionCol: 'p'},
...     {lr.regParam: 2.0, lr.maxIter: 1, lr.labelCol: 'l', lr.predictionCol: 'p'},
...     {lr.regParam: 1.0, lr.maxIter: 5, lr.labelCol: 'l', lr.predictionCol: 'p'},
...     {lr.regParam: 2.0, lr.maxIter: 5, lr.labelCol: 'l', lr.predictionCol: 'p'}]
>>> len(output) == len(expected)
True
>>> all([m in expected for m in output])
True
```

New in version 1.4.0.

addGrid(*param*, *values*) [source]

Sets the given parameters in this grid to fixed values.

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*) [source]

K-fold cross validation.

```
>>> from pyspark.ml.classification import LogisticRegression
>>> from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>> from pyspark.mllib.linalg import Vectors
>>> dataset = sqlContext.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')*

param for estimator to be cross-validated

estimatorParamMaps = *Param(parent='undefined', name='estimatorParamMaps', doc='estimator param maps')*

param for estimator param maps

evaluator = *Param(parent='undefined', name='evaluator', doc='evaluator used to select hyper-parameters that maximize the cross-validated metric')*

param for the evaluator used to select hyper-parameters that maximize the cross-validated 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() [source]

Gets the value of estimator or its default value.

New in version 1.4.0.

getEstimatorParamMaps() [source]

Gets the value of estimatorParamMaps or its default value.

New in version 1.4.0.

getEvaluator() [source]

Gets the value of evaluator or its default value.

New in version 1.4.0.

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.

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')*

param for 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.

setEstimator(*value*) [source]

Sets the value of **estimator**.

New in version 1.4.0.

setEstimatorParamMaps(*value*) [source]

Sets the value of **estimatorParamMaps**.

New in version 1.4.0.

setEvaluator(*value*) [source]

Sets the value of **evaluator**.

New in version 1.4.0.

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): Sets params for cross validator.

New in version 1.4.0.

class pyspark.ml.tuning.**CrossValidatorModel**(*bestModel*) [source]

Model from k-fold cross validation.

New in version 1.4.0.

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.

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.

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]

Evaluator for binary classification, which expects two input columns: rawPrediction and label.

```
>>> from pyspark.mllib.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 = sqlContext.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)')*

param for 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]

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 = sqlContext.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 (mse|rmse|r2|mae)')*

param for metric name in evaluation (mse|rmse|r2|mae)

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]

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 = sqlContext.createDataFrame(scoreAndLabels, [“prediction”, “label”]) ... >>> evaluator = MulticlassClassificationEvaluator(predictionCol=”prediction”) >>> evaluator.evaluate(dataset) 0.66... >>> evaluator.evaluate(dataset, {evaluator.metricName: “precision”}) 0.66... >>> evaluator.evaluate(dataset, {evaluator.metricName: “recall”}) 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|precision|recall|weightedPrecision|weightedRecall)')*

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**.