

Announcing Stack Overflow Documentation

We started with Q&A. Technical documentation is next, and we need your help.

Whether you're a beginner or an experienced developer, you *can* contribute.

[Sign up and start helping →](#)[Learn more about Documentation →](#)

Create a Python transformer on sparsevector data type column in Pyspark ML



I have a dataframe with a column *'features'* (each row in the dataframe represents a document). I used HashingTF to calculate the column *'tf'* and I also created a custom transformer *'TermCount'* (just as test) to calculate the *'total_terms'* as follows:

```
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row
from pyspark.ml.pipeline import Transformer
from pyspark.ml.param.shared import HasInputCol, HasOutputCol, Param
from pyspark.ml.feature import HashingTF
from pyspark.ml.util import keyword_only
from pyspark.mllib.linalg import SparseVector
from pyspark.sql.functions import udf
```

```
class TermCount(Transformer, HasInputCol, HasOutputCol):
```

```
    @keyword_only
```

```
    def __init__(self, inputCol=None, outputCol=None):
        super(TermCount, self).__init__()
```

```

kwargs = self.__init__._input_kwargs
self.setParams(**kwargs)

@keyword_only
def setParams(self, inputCol=None, outputCol=None):
    kwargs = self.setParams._input_kwargs
    return self._set(**kwargs)

def _transform(self, dataset):

    def f(s):
        return len(s.values)

    out_col = self.getOutputCol()
    in_col = dataset[self.getInputCol()]
    return dataset.withColumn(out_col, udf(f)(in_col))

sc = SparkContext()
sqlContext = SQLContext(sc)
documents = sqlContext.createDataFrame([
    (0, "w1 w2 w3 w4 w1 w1 w1"),
    (1, "w2 w3 w4 w2"),
    (2, "w3 w4 w3"),
    (3, "w4")], ["doc_id", "doc_text"])

df = documents.map(lambda x : (x.doc_id,x.doc_text.split("
"))).toDF().withColumnRenamed("_1","doc_id").withColumnRenamed("_2","features")

htf = HashingTF(inputCol="features", outputCol="tf")
tf = htf.transform(df)

term_count_model=TermCount(inputCol="tf", outputCol="total_terms")
tc_df=term_count_model.transform(tf)
tc_df.show(truncate=False)
#+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
#-----+
#|doc_id|features                                |tf
#|total_terms|
#+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
#-----+
#|0      |[w1, w2, w3, w4, w1, w1, w1]|(262144,[3738,3739,3740,3741],[4.0,1.0,1.0,1.0])|4
#|
#|1      |[w2, w3, w4, w2]          |(262144,[3739,3740,3741],[2.0,1.0,1.0])          |3
#|
#|2      |[w3, w4, w3]              |(262144,[3740,3741],[2.0,1.0])                    |2
#|
#|3      |[w4]                      |(262144,[3741],[1.0])                             |1
#|
#+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

-----+

Now, I need to add a similar transformer which receives 'tf' as an inputCol and compute the document frequency for each term (no_of_rows_contains_this_term / total_no_of_rows) to an outputCol of type Sparsevector and finally to get a result like this:

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|doc_id|features          |tf
|total_terms| doc_freq          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|0      |[w1, w2, w3, w4, w1, w1, w1]|(262144,[3738,3739,3740,3741],[4.0,1.0,1.0,1.0])|4
|(262144,[3738,3739,3740,3741],[0.25,0.50,0.75,1.0])|
|1      |[w2, w3, w4, w2]|(262144,[3739,3740,3741],[2.0,1.0,1.0])|3
|(262144,[3739,3740,3741],[0.50,0.75,1.0])|
|2      |[w3, w4, w3]|(262144,[3740,3741],[2.0,1.0])|2
|(262144,[3740,3741],[0.75,1.0])|
|3      |[w4]|(262144,[3741],[1.0])|1
|(262144,[3741],[1.0])|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

python pyspark apache-spark-mllib

asked Mar 9 at 18:13



K.Ali

43 6

I used @zero323 's idea of [Python transformaer](#) – [K.Ali](#) Mar 10 at 9:14

1 Answer

Excluding all the wrapping code you can try to use `Statistics.colStats` :

```
from pyspark.mllib.stat import Statistics
from pyspark.mllib.linalg import Vectors
```

```
tf_col = "x"
dataset = sc.parallelize([
    "(262144,[3738,3739,3740,3741],[0.25,0.50,0.75,1.0])",
```

```

"(262144,[3738,3739,3740,3741],[0.25,0.50,0.75,1.0])"
])).map(lambda s: (Vectors.parse(s),)).toDF(["x"])

vs = (dataset.select(tf_col)
      .flatMap(lambda x: x)
      .map(lambda v: Vectors.sparse(v.size, v.indices, [1.0 for _ in v.values])))

stats = Statistics.colStats(vs)

document_frequency = stats.mean()
document_frequency.max()
## 1.0
document_frequency.min()
# 0.0
document_frequency.nonzero()
## (array([3738, 3739, 3740, 3741]),)

```

When you have this information you can easily adjust required indices:

```

from pyspark.mllib.linalg import VectorUDT

df = Vectors.sparse(
    document_frequency.shape[0], document_frequency.nonzero()[0],
    document_frequency[document_frequency.nonzero()]
)

def idf(df, d):
    values = ... # Compute new values
    return Vectors.sparse(v.size, v.indices, values)

dataset.withColumn("idf_col", udf(idf, VectorUDT()(col("tf_col"))))

```

A huge caveat is that `stats.mean` returns a `DenseVector` so if you have TF with 262144 features the output is an array of the same length.

edited Mar 10 at 20:58

answered Mar 10 at 13:56



[zero323](#)

66.1k 16 77 137

All you need is an udf here. – [zero323](#) Mar 10 at 21:02

Oh thanx @zero323 , that's kind of you. `dataset` did not change after calling `idf()` function. still contains on column rather than two. – [K.Ali](#) Mar 11 at 19:02
