Categorical Features

Lesson Objectives

- After completing this lesson, you should be able to:
  - encode categorical features with Spark's `StringIndexer`
  - encode categorical features with Spark's `OneHotEncoder`
  - know when to use each of these

Motivation

- Categorical variables can take on only a limited number of possible values, like country, or gender
- They represent reality. You don’t have infinite variation in between two countries. You do have infinite values between two integers
- Categories are less useful than integers for computations. So internally a computer will ‘translate’ categorical variables to integers

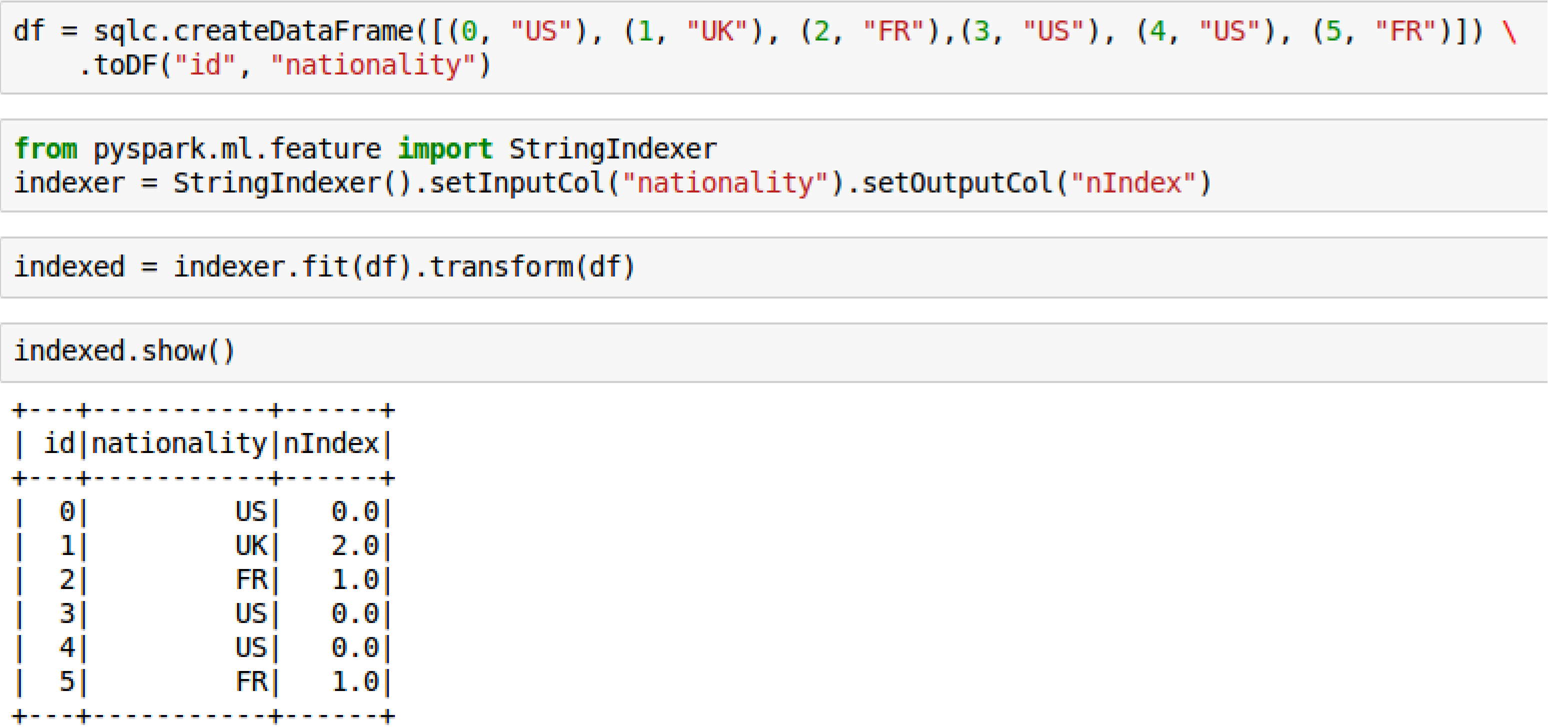
Motivation

- In `R` you have `factors`
- In python pandas you have the `categorical` data type. What is the equivalent data structure in Spark?
- These structures usually map strings to integers in a way that makes future computations easier. In this video we will see how Spark does it

Why are integers better?

- Spark’s classifiers and regressors only work with numerical features; string features must be converted to numbers a `StringIndexer`
- This helps keep Spark’s internals simpler and more efficient
- There’s little cost in transforming categorical features to numbers, and then back to strings

Using a `StringIndexer`



```
df = sqlc.createDataFrame([(0, "US"), (1, "UK"), (2, "FR"),(3, "US"), (4, "US"), (5, "FR")]) \
    .toDF("id", "nationality")
```

```
from pyspark.ml.feature import StringIndexer
indexer = StringIndexer().setInputCol("nationality").setOutputCol("nIndex")
```

```
indexed = indexer.fit(df).transform(df)
```

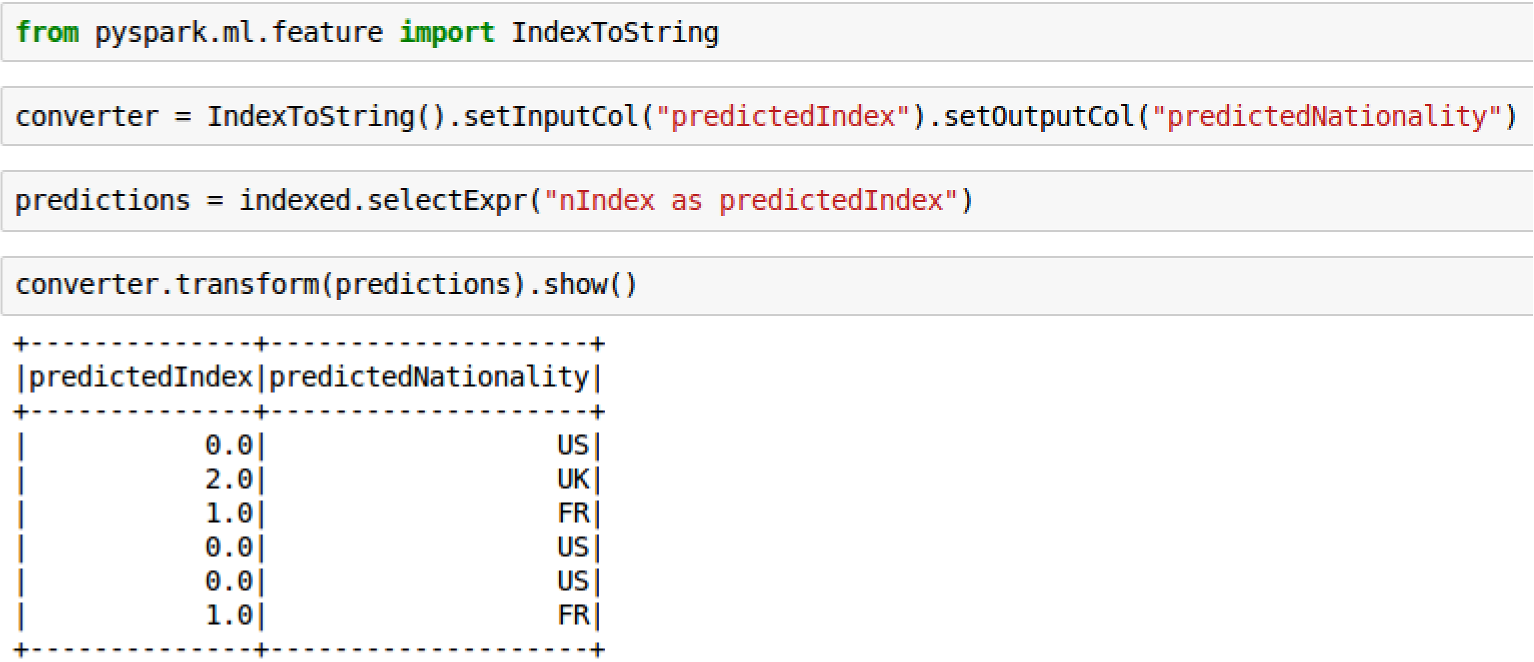
```
indexed.show()
```

```
+---+-----------+------+
| id|nationality|nIndex|
+---+-----------+------+
|  0|         US|   0.0|
|  1|         UK|   2.0|
|  2|         FR|   1.0|
|  3|         US|   0.0|
|  4|         US|   0.0|
|  5|         FR|   1.0|
+---+-----------+------+
```

Reversing the Mapping

- The classifiers in MLlib and spark.ml will predict numeric values that correspond to the index values
- `IndexToString` is what you’ll need to transform these numbers back into your original labels

IndexToString example



```
from pyspark.ml.feature import IndexToString
```

```
converter = IndexToString().setInputCol("predictedIndex").setOutputCol("predictedNationality")
```

```
predictions = indexed.selectExpr("nIndex as predictedIndex")
```

```
converter.transform(predictions).show()
```

```
+--------------+--------------------+
|predictedIndex|predictedNationality|
+--------------+--------------------+
|           0.0|                  US|
|           2.0|                  UK|
|           1.0|                  FR|
|           0.0|                  US|
|           0.0|                  US|
|           1.0|                  FR|
+--------------+--------------------+
```

One Hot Encoding

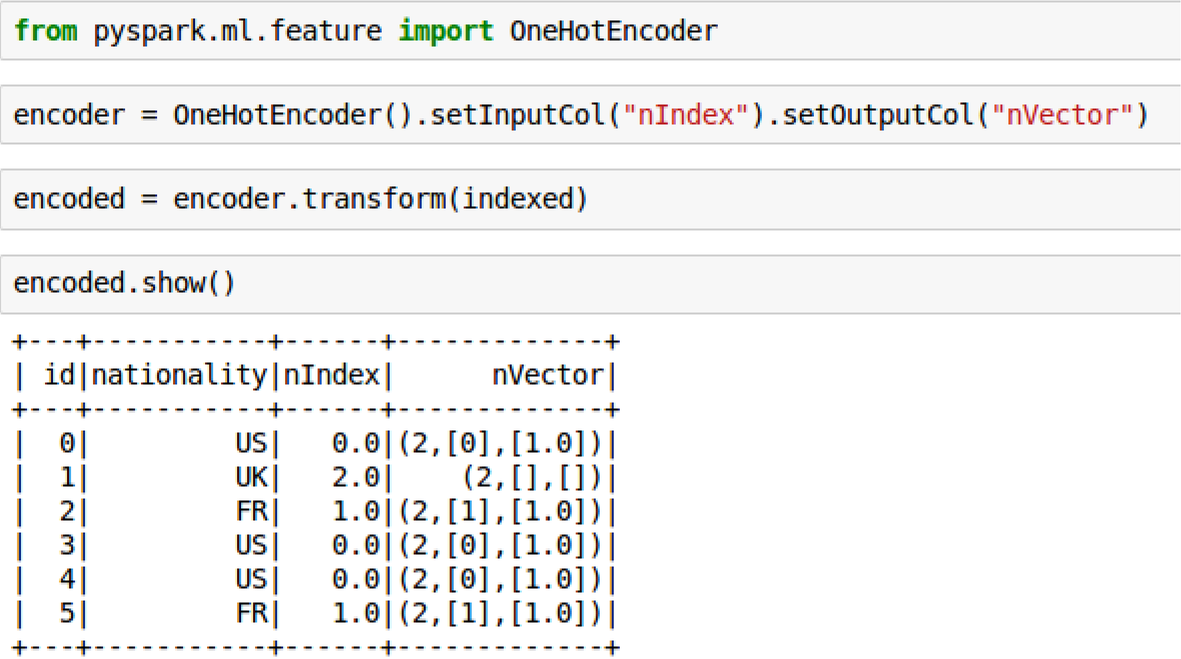
- Suppose we are trying to fit a linear regressor that uses nationality as a feature
- It would be impossible to learn a weight for this one feature that can distinguish between the 3 nationalities in our dataset
- It’s better to instead have a separate Boolean feature for each nationality, and learn weights for those features independently

Spark’s `OneHotEncoder`

- The `OneHotEncoder` creates a sparse vector column, with each dimension of this vector of Booleans representing one of the possible values of the original feature

nationality	**encoding: [US, FR, UK]**
"US"	[1,0,0]
"UK"	[0,0,1]
"FR"	[0,1,0]
"US"	[1,0,0]
"US"	[1,0,0]
"FR"	[0,1,0]

Using a OneHotEncoder



```
from pyspark.ml.feature import OneHotEncoder
```

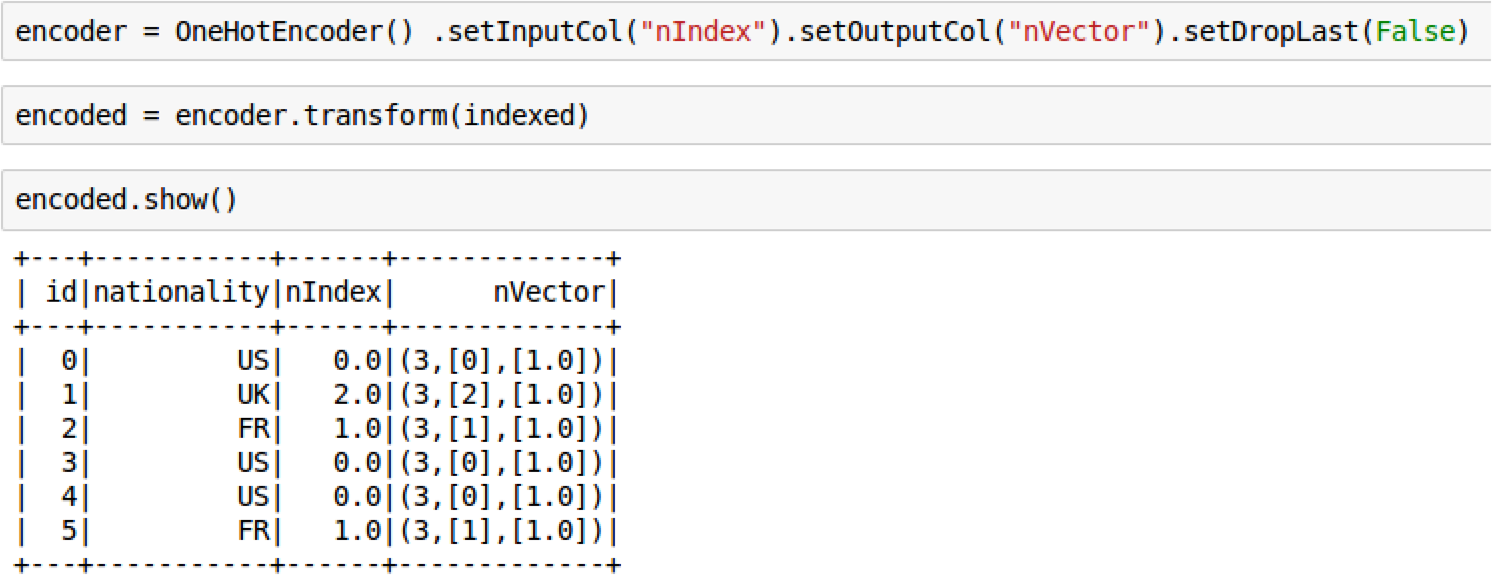
```
encoder = OneHotEncoder().setInputCol("nIndex").setOutputCol("nVector")
```

```
encoded = encoder.transform(indexed)
```

```
encoded.show()
```

```
+---+-----------+------+-------------+
| id|nationality|nIndex|      nVector|
+---+-----------+------+-------------+
|  0|         US|   0.0|(2,[0],[1.0])|
|  1|         UK|   2.0|    (2,[],[])|
|  2|         FR|   1.0|(2,[1],[1.0])|
|  3|         US|   0.0|(2,[0],[1.0])|
|  4|         US|   0.0|(2,[0],[1.0])|
|  5|         FR|   1.0|(2,[1],[1.0])|
+---+-----------+------+-------------+
```

The `dropLast` Option



```
encoder = OneHotEncoder() .setInputCol("nIndex").setOutputCol("nVector").setDropLast(False)
```

```
encoded = encoder.transform(indexed)
```

```
encoded.show()
```

```
+---+-----------+------+-------------+
| id|nationality|nIndex|      nVector|
+---+-----------+------+-------------+
|  0|         US|   0.0|(3,[0],[1.0])|
|  1|         UK|   2.0|(3,[2],[1.0])|
|  2|         FR|   1.0|(3,[1],[1.0])|
|  3|         US|   0.0|(3,[0],[1.0])|
|  4|         US|   0.0|(3,[0],[1.0])|
|  5|         FR|   1.0|(3,[1],[1.0])|
+---+-----------+------+-------------+
```

Lesson Summary

- Having completed this lesson, you should be able to:
  - encode categorical features with Spark's `StringIndexer`
  - encode categorical features with Spark's `OneHotEncoder`
  - know when to use each of these