Chapter 7: Logistic Regression - Pipeline

Ex2: Titanic

Dataset 'titanic.csv'.

Requirement:

- Read data
- Pre-process data.
- With some information: 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked' => build a model (use Pipeline) to predict if a passenger on Titanic 'Survived' or not
- Estimate this model.

In [1]:

```
import findspark
findspark.init()
```

In [2]:

```
from pyspark.sql import SparkSession
```

In [3]:

```
spark = SparkSession.builder.appName('myproj').getOrCreate()
```

In [4]:

```
data = spark.read.csv('titanic.csv',inferSchema=True,header=True)
```

In [5]:

```
data.count()
```

Out[5]:

```
891
```

In [6]:

```
data.printSchema()
```

```
root
 |-- PassengerId: integer (nullable = true)
 |-- Survived: integer (nullable = true)
 |-- Pclass: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- SibSp: integer (nullable = true)
 |-- Parch: integer (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: double (nullable = true)
 |-- Cabin: string (nullable = true)
 |-- Embarked: string (nullable = true)
```

```
In [7]: data.columns
```

```
Out[7]: ['PassengerId',
         'Survived',
         'Pclass',
         'Name',
         'Sex',
         'Age',
         'SibSp',
         'Parch',
         'Ticket',
         'Fare',
         'Cabin',
         'Embarked']
```

```
In [8]: my_cols = data.select(['Survived',
         'Pclass',
         'Sex',
         'Age',
         'SibSp',
         'Parch',
         'Fare',
         'Embarked'])
```

```
In [9]: my_final_data = my_cols.na.drop()
```

Working with Categorical Columns

Let's break this down into multiple steps to make it all clear.

```
In [10]: from pyspark.ml.feature import (VectorAssembler,VectorIndexer,
                                         OneHotEncoder,StringIndexer)
```

```
In [11]: gender_indexer = StringIndexer(inputCol='Sex',
                                        outputCol='SexIndex')
         gender_encoder = OneHotEncoder(inputCol='SexIndex',
                                        outputCol='SexVec')
```

```
In [12]: embark_indexer = StringIndexer(inputCol='Embarked',
                                        outputCol='EmbarkIndex')
         embark_encoder = OneHotEncoder(inputCol='EmbarkIndex',
                                        outputCol='EmbarkVec')
```

```
In [13]: assembler = VectorAssembler(inputCols=['Pclass',
          'SexVec',
          'Age',
          'SibSp',
          'Parch',
          'Fare',
          'EmbarkVec'],outputCol='features')
```

In [14]:
```
from pyspark.ml.classification import LogisticRegression
```

Pipelines

Let's see an example of how to use pipelines (we'll get a lot more practice with these later!)

In [15]:
```
from pyspark.ml import Pipeline
```

In [16]:
```
log_reg_titanic = LogisticRegression(featuresCol='features',
                                     labelCol='Survived')
```

In [17]:
```
pipeline = Pipeline(stages=[gender_indexer,embark_indexer,
                           gender_encoder,embark_encoder,
                           assembler,log_reg_titanic])
```

In [18]:
```
train_titanic_data, test_titanic_data = my_final_data.randomSplit([0.7,.3])
```

In [19]:
```
fit_model = pipeline.fit(train_titanic_data)
```

In [20]:
```
results = fit_model.transform(test_titanic_data)
```

In [21]:
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

In [22]:
```
my_eval = BinaryClassificationEvaluator(rawPredictionCol='prediction',
                                        labelCol='Survived')
```

In [23]:
```
results.select('Survived','prediction').show()
```

```
+--------+----------+
|Survived|prediction|
+--------+----------+
|       0|       1.0|
|       0|       1.0|
|       0|       1.0|
|       0|       1.0|
|       0|       1.0|
|       0|       0.0|
|       0|       0.0|
|       0|       1.0|
|       0|       0.0|
|       0|       1.0|
|       0|       0.0|
|       0|       0.0|
|       0|       0.0|
|       0|       1.0|
|       0|       0.0|
|       0|       0.0|
|       0|       0.0|
|       0|       0.0|
|       0|       0.0|
|       0|       1.0|
+--------+----------+
only showing top 20 rows
```

In [24]:
```
AUC = my_eval.evaluate(results)
```

In [25]:
```
AUC
```

Out[25]:
```
0.7974789915966386
```