Testing MLlib with Python 3 in Docker (Advanced)

Goals

In a related notebook we did simple regression with one numeric. In this case, I want to do the following:

- Add a second categorical variable that I will encode (e.g. zipcode)
- Find way to get model evaluation metrics
- Save and reload the ML pipeline
- Expiriment with updating the parameters of the pipeline without re-initializing it.

In [2]:
```
from pyspark.sql import SparkSession

spark = SparkSession\
    .builder\
    .appName("PythonMLlib")\
    .getOrCreate()
```

Importing Libraries

Notes on libraries:

- Use VectorAssembler to convert df columns to features column. Notes here (http://stackoverflow.com/questions/37574833/create-labledpoints-from-spark-dataframe-how-to-pass-list-of-names-to-vectoras).
- Pipeline example with regression here (https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3229648579127/4041455208 utm_campaign=2016%20Spark%20Summit%20West&utm_medium=social&utm_source=slide).
- We used cast and withColumn to convert the column types. Cannot directly assign dataframe column. Sample code here (http://stackoverflow.com/questions/32284620/how-to-change-a-dataframe-column-from-string-type-to-double-type-in-pyspark)
- We used the StringIndexer and OHE encode categorical variables(e.g. zipcode). Sample code here (https://spark.apache.org/docs/latest/ml-features.html#onehotencoder). The encoded column looks weird but it works. It's a touple in some odd format.

In [6]:
```
from pyspark.sql.types import DoubleType, IntegerType
from pyspark.ml.feature import VectorAssembler, OneHotEncoder, StringIndexer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.regression import GeneralizedLinearRegression
```

Data munging

In [10]:
```
df = spark.read.csv('home_data.csv', header=True)
df = df.withColumn("price", df["price"].cast(DoubleType()))\
                   .withColumn("sqft_living", df["sqft_living"].cast(DoubleTyp
e()))

df.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- date: string (nullable = true)
 |-- price: double (nullable = true)
 |-- bedrooms: string (nullable = true)
 |-- bathrooms: string (nullable = true)
 |-- sqft_living: double (nullable = true)
 |-- sqft_lot: string (nullable = true)
 |-- floors: string (nullable = true)
 |-- waterfront: string (nullable = true)
 |-- view: string (nullable = true)
 |-- condition: string (nullable = true)
 |-- grade: string (nullable = true)
 |-- sqft_above: string (nullable = true)
 |-- sqft_basement: string (nullable = true)
 |-- yr_built: string (nullable = true)
 |-- yr_renovated: string (nullable = true)
 |-- zipcode: string (nullable = true)
 |-- lat: string (nullable = true)
 |-- long: string (nullable = true)
 |-- sqft_living15: string (nullable = true)
 |-- sqft_lot15: string (nullable = true)
```

In [11]:
```
(trainData, testData) = df.randomSplit(seed=123, weights=[0.7,0.3])
print("The total data is {}, the training is {} and the test is {}"\
      .format(df.count(), trainData.count(), testData.count()))
```

```
The total data is 21613, the training is 15089 and the test is 6524
```

Train & Evaluate Model

Notes:

- Model evaluation values like p-value from model summary object. Code sample here (https://home.apache.org/~pwendell/spark-nightly/spark-branch-2.0-docs/latest/ml-classification-regression.html#generalized-linear-regression)
- Prediction values come from an evaluator object. Code sample here (https://home.apache.org/~pwendell/spark-nightly/spark-branch-2.0-docs/latest/ml-classification-regression.html#decision-tree-regression)

In [12]:
```
stringifier = StringIndexer(inputCol="zipcode", outputCol="zipIndex")
oneHotter = OneHotEncoder(inputCol="zipIndex", outputCol="zipVector")
vectorizer = VectorAssembler(inputCols=["sqft_living", "zipVector"], outputCol
eatures")
glr = GeneralizedLinearRegression(labelCol="price", family="gaussian", link="i
dentity", maxIter=10, regParam=0.3)
simplePipeline = Pipeline(stages=[stringifier, oneHotter, vectorizer, glr])
model = simplePipeline.fit(trainData)


# Summarize the model over the training set and print out some metrics
#print("AIC: " + str(model.aic))
#print("Coefficient Standard Errors: " + str(model.coefficientStandardErrors))
#print("T Values: " + str(model.tValues))
#print("P Values: " + str(model.pValues))
#print("Deviance Residuals: ")
#model.residuals
```

In [13]:
```
#testingData = vectorizer.transform(testData)
# Make predictions.
predictions = model.transform(testData)

# Select example rows to display.
predictions.select("prediction", "price", "features").show(5)

# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(labelCol="price", predictionCol="prediction",
metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("RMSE is: {}".format(rmse))
```

```
+------------------+--------+--------------------+
|        prediction|   price|            features|
+------------------+--------+--------------------+
| 424939.7035272836|300000.0|(70,[0,54],[2400....|
|462071.42308817804|647500.0|(70,[0,48],[2060....|
| 311486.7681929148|400000.0|(70,[0,48],[1460....|
| 556784.1148660106|487000.0|(70,[0,57],[2540....|
|  359747.505235619|239000.0|(70,[0,42],[1980....|
+------------------+--------+--------------------+
only showing top 5 rows

RMSE is: 193402.05997401488
```

We'll save and load the pipeline we just executed

In [19]:
```
simplePipeline.save("glmPipelinewithEncoding.ml")
retrievedPipeline = Pipeline()
retrievedPipeline.load("glmPipelinewithEncoding.ml")
print(retrievedModel is None)
```

```
False
```

Let's now re-run the prediction to see if it works

In [16]:
```
retrievedPreds = retrievedPipeline.fit(trainData).transform(testData)
print("RMSE from preds is: {} and should be same as: {}"\
      .format(RegressionEvaluator(labelCol="price",
predictionCol="prediction", metricName="rmse"), rmse))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-379ffb240891> in <module>()
----> 1 retrievedPreds = retrievedPipeline.fit(trainData).transform(testData)
      2 print("RMSE from preds is: {} and should be same as: {}"      .format(
ressionEvaluator(labelCol="price", predictionCol="prediction", metricName="rm
se"), rmse))

/usr/local/spark/python/pyspark/ml/base.py in fit(self, dataset, params)
     62                 return self.copy(params)._fit(dataset)
     63             else:
---> 64                 return self._fit(dataset)
     65         else:
     66             raise ValueError("Params must be either a param map or a
 list/tuple of param maps, "

/usr/local/spark/python/pyspark/ml/pipeline.py in _fit(self, dataset)
     96     def _fit(self, dataset):
     97         stages = self.getStages()
---> 98         for stage in stages:
     99             if not (isinstance(stage, Estimator) or isinstance(stage,
Transformer)):
    100                 raise TypeError(

TypeError: 'NoneType' object is not iterable
```

In [20]:
```
testData is None
```

Out[20]:
```
False
```