cal_housing_CH

February 19, 2021

0.0.1 Linear Regression Modeling of California Home Prices

0.0.2 University of California, Santa Barbara

0.0.3 PSTAT 135/235: Big Data Analytics

0.0.4 Last Updated: May 30, 2020

TOTAL POINTS: 10

Instructions
In this project, you will work with the California Home Price dataset to train a regression model and predict median home prices. Please do the following:

1) (6 PTS) Go through all code and fill in the missing cells. This will prep data, train a model, predict, and evaluate model fit. Compute and report the Mean Squared Error (MSE).

2) (1 PT) Repeat Part 1 with at least one additional feature from the original set.

3) (2 PTS) Repeat Part 1 with at least one engineered feature based on one or more variables from the original set.

4) (1 PT) Repeat Part 1 using Lasso Regression

Please report resuts in the following way:
In the **RESULTS SECTION** table at the very bottom, there are three cells where you should copy your code from parts 2,3,4.
In the very last cell, print a dataframe containing two columns: `question_part` and `MSE`.
This dataframe must report your MSE results.

Data Source
StatLib—Datasets Archive
http://lib.stat.cmu.edu/datasets/

[1]:
```
import os
import pandas as pd

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
```

[2]:
```
# read text file into pyspark dataframe
filename = 'cal_housing_data_preproc_w_header.txt'
df = spark.read.csv(filename,  inferSchema=True, header = True)
```

[3]:
```
df.show(3)
```

```
+------------------+-----------------+------------------+-----------+-----------
---+----------+----------+--------+---------+
|median_house_value|    median_income|housing_median_age|total_rooms|total_bedro
oms|population|households|latitude|longitude|
+------------------+-----------------+------------------+-----------+-----------
---+----------+----------+--------+---------+
|          452600.0|           8.3252|              41.0|      880.0|
129.0|     322.0|     126.0|   37.88|  -122.23|
|          358500.0|           8.3014|              21.0|     7099.0|
1106.0|    2401.0|    1138.0|   37.86|  -122.22|
|          352100.0|7.257399999999999|              52.0|     1467.0|
190.0|     496.0|     177.0|   37.85|  -122.24|
+------------------+-----------------+------------------+-----------+-----------
---+----------+----------+--------+---------+
only showing top 3 rows
```

0.0.5 Additional Preprocessing

We want to do three more things before training a model:

SCALING (1 POINT)
Scale the response variable median_house_value, dividing by 100000 and saving into column median_house_value_final

[4]:
```
df_scaled = df.withColumn("median_house_value_final", df.median_house_value /
                         100000)
```

FEATURE ENGINEERING (1 POINT)
Add new feature: rooms_per_household

[5]:
```
df2 = df_scaled.withColumn("rooms_per_household", df.total_rooms / df.
 ↪households)
```

SELECT AND STANDARDIZE FEATURES (2 POINTS)

[7]:
```
# retain these predictors for Part 1
vars_to_keep = ["median_house_value_final",
              "total_bedrooms",
              "population",
              "households",
              "median_income",
               "rooms_per_household"]


from pyspark.ml.feature import VectorAssembler

# subset the dataframe on these predictors

assembler = VectorAssembler(inputCols = ["total_bedrooms", "population",␣
 ↪"households", "median_income", "rooms_per_household"], outputCol =␣
 ↪"features")
transformed = assembler.transform(df2)
```

[12]:

```
from pyspark.ml.linalg import DenseVector
#RDD
dataRdd = transformed.select(transformed.median_house_value_final, transformed.
 ↪features).rdd.map(tuple)
data_df = dataRdd.toDF()
data_df = data_df.withColumnRenamed("_1", "label").withColumnRenamed("_2",␣
 ↪"features")
data_df.show(5)
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|4.526|[129.0,322.0,126...|
|3.585|[1106.0,2401.0,11...|
|3.521|[190.0,496.0,177...|
|3.413|[235.0,558.0,219...|
|3.422|[280.0,565.0,259...|
+-----+--------------------+
only showing top 5 rows
```

We want to standardize the features, but not the response variable.

[14]:

```
from pyspark.ml.feature import StandardScaler
```

[16]:

```
# Feature scaling

# Initialize the `standardScaler`
standardScaler = StandardScaler(inputCol="features",␣
 ↪outputCol="features_scaled",
                                withStd=True, withMean=False)

# Fit the DataFrame to the scaler; this computes the mean, standard deviation␣
 ↪of each feature
scaler = standardScaler.fit(transformed)
```

```
# Transform the data in `df2` with the scaler
scaled_df = scaler.transform(transformed)
```

Split data into train set (80%), test set (20%) using seed=314

[18]:
```
seed = 314
train_test = [0.8, 0.2]
train_data, test_data = scaled_df.randomSplit(train_test,seed)
```

Initialize the linear regression object with given parameters **(1 POINT)**

[19]:
```
from pyspark.ml.regression import LinearRegression # note this is from the ML␣
 ↪package

maxIter=10
regParam=0.3
elasticNetParam=0.8

lr = LinearRegression(featuresCol = "features_scaled", labelCol =␣
 ↪"median_house_value_final", maxIter = 10, elasticNetParam = 0.8, regParam =␣
 ↪0.3)
```

Fit the model using the training data

[20]:
```
model = lr.fit(train_data)
```

For each datapoint in the test set, make a prediction (hint: apply `transform()` to the model). You will want the returned object to be a dataframe

[22]:
```
prediction = model.transform(test_data)
```

COMPUTE MSE (1 POINT)

Evaluate the model by computing Mean Squared Error (MSE), which is the average sum of squared differences between predicted and label.

This can be computed in a single line using `reduce()`

[26]:
```
from pyspark.ml.evaluation import RegressionEvaluator
eval = RegressionEvaluator(labelCol = "median_house_value_final", predictionCol␣
 ↪= "prediction", metricName = "rmse")
mse = eval.evaluate(prediction, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)
```

```
MSE: 0.755
```

RESULTS SECTION

[28]:
```
# Code for Part 2
```

```
from pyspark.ml.feature import VectorAssembler
# subset the dataframe on these predictors

assembler2 = VectorAssembler(inputCols = ["total_bedrooms", "population",␣
 ↪"households", "median_income", "rooms_per_household", "housing_median_age"],␣
 ↪outputCol = "features")
transformed2 = assembler2.transform(df2)
standardScaler2 = StandardScaler(inputCol="features",␣
 ↪outputCol="features_scaled",
                                withStd=True, withMean=False)

# Fit the DataFrame to the scaler; this computes the mean, standard deviation␣
 ↪of each feature
scaler2 = standardScaler2.fit(transformed2)

# Transform the data in `df2` with the scaler
scaled_df2 = scaler2.transform(transformed2)

train_data2, test_data2 = scaled_df2.randomSplit(train_test,seed)
lr = LinearRegression(featuresCol = "features_scaled", labelCol =␣
 ↪"median_house_value_final", maxIter = 10, elasticNetParam = 0.8, regParam =␣
 ↪0.3)

model2 = lr.fit(train_data2)
prediction2 = model2.transform(test_data2)

eval = RegressionEvaluator(labelCol = "median_house_value_final", predictionCol␣
 ↪= "prediction", metricName = "rmse")
mse2 = eval.evaluate(prediction2, {eval.metricName: "mse"})
print("MSE: %.3f" % mse2)
```

```
MSE: 0.755
```

[30]:

```
# Code for Part 3

df3 = df2.withColumn("households_per_pop", (df2.households / df2.population))

assembler3 = VectorAssembler(inputCols = ["total_bedrooms", "population",␣
 ↪"households", "median_income", "rooms_per_household", "housing_median_age",␣
 ↪"households_per_pop"], outputCol = "features")
transformed3 = assembler3.transform(df3)
standardScaler3 = StandardScaler(inputCol="features",␣
 ↪outputCol="features_scaled",
                                withStd=True, withMean=False)

# Fit the DataFrame to the scaler; this computes the mean, standard deviation␣
 ↪of each feature
```

```
scaler3 = standardScaler3.fit(transformed3)

# Transform the data in `df2` with the scaler
scaled_df3 = scaler3.transform(transformed3)

train_data3, test_data3 = scaled_df3.randomSplit(train_test,seed)

lr3 = LinearRegression(featuresCol = "features_scaled", labelCol =␣
 ↪"median_house_value_final", maxIter = 10, elasticNetParam = 0.8, regParam =␣
 ↪0.3)

model3 = lr.fit(train_data3)
prediction3 = model3.transform(test_data3)

#calculate MSE
mse3 = eval.evaluate(prediction3, {eval.metricName: "mse"})
print("MSE: %.3f" % mse3)
```

```
MSE: 0.719
```

[31]:

```
# Code for Part 4

lr4 = LinearRegression(featuresCol = "features_scaled", labelCol =␣
 ↪"median_house_value_final", maxIter = 10, elasticNetParam = 1.0, regParam =␣
 ↪0.3)

model4 = lr4.fit(train_data)
prediction4 = model4.transform(test_data)

eval = RegressionEvaluator(labelCol = "median_house_value_final", predictionCol␣
 ↪= "prediction", metricName = "rmse")

mse4 = eval.evaluate(prediction4, {eval.metricName: "mse"})
print("MSE: %.3f" % mse4)
```

```
MSE: 0.775
```

Print dataframe containing `question_part`, `MSE` values for parts 1-4 in the next cell.

[32]:

```
# print dataframe containing question_part, MSE
anwser = spark.createDataFrame(
    [
        (1, mse),
        (2, mse2),
        (3, mse3),
        (4, mse4)
    ],
    ["question_part", "MSE"]
)
anwser.show()
```

```
+-------------+------------------+
|question_part|               MSE|
+-------------+------------------+
|            1|0.7551749809615158|
|            2|0.7551747847051158|
|            3|0.7185818220171625|
|            4|0.7746803974273968|
+-------------+------------------+
```

[ ]:

```
# Save notebook as PDF document
!jupyter nbconvert --to pdf `pwd`/*.ipynb
```