University of Virginia

DS 5559: Big Data Analytics

Linear Regression Modeling of California Home Prices

Last updated: Oct 21, 2019

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/

In [80]:
```
import os
import pandas as pd

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

In [81]:
```
# read text file into pyspark dataframe
filename = 'cal_housing_data_preproc_w_header.txt'
df = spark.read.csv(filename,  inferSchema=True, header = True)
```

In [82]:
```
df.show(3)
```

```
+------------------+-----------------+------------------+-----------+-
-------------+----------+----------+--------+---------+
|median_house_value|    median_income|housing_median_age|total_rooms|total_
```

```
bedrooms|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
```

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

In [83]:
```
from pyspark.sql.functions import col

df = df.withColumn("median_house_value_final", df.median_house_value/10000
0)
df.show(3)
```

```
+-----------------+----------------+-----------------+-----------+-
-------------+----------+----------+--------+---------+--------------
---------+
|median_house_value|    median_income|housing_median_age|total_rooms|total_
bedrooms|population|households|latitude|longitude|median_house_value_final|
+-----------------+----------------+-----------------+-----------+-
-------------+----------+----------+--------+---------+--------------
---------+
|          452600.0|          8.3252|             41.0|      880.0|
129.0|     322.0|     126.0|   37.88|  -122.23|                 4.526|
|          358500.0|          8.3014|             21.0|     7099.0|
1106.0|    2401.0|    1138.0|   37.86|  -122.22|                 3.585|
|          352100.0|7.257399999999999|             52.0|     1467.0|
190.0|     496.0|     177.0|   37.85|  -122.24|                 3.521|
+-----------------+----------------+-----------------+-----------+-
-------------+----------+----------+--------+---------+--------------
---------+
only showing top 3 rows
```

FEATURE ENGINEERING (1 POINT)

Add new feature: rooms_per_household

In [84]:
```
df = df.withColumn('rooms_per_household', df.total_rooms / df.households)
df.show(3)
```

```
+------------------+-----------------+------------------+-----------+-
------------+----------+----------+--------+---------+--------------
---------+-------------------+
|median_house_value|    median_income|housing_median_age|total_rooms|total_
bedrooms|population|households|latitude|longitude|median_house_value_final|
rooms_per_household|
+------------------+-----------------+------------------+-----------+-
------------+----------+----------+--------+---------+--------------
---------+-------------------+
|          452600.0|           8.3252|              41.0|      880.0|
129.0|     322.0|     126.0|   37.88|  -122.23|                   4.526|  6.
984126984126984|
|          358500.0|           8.3014|              21.0|     7099.0|
1106.0|    2401.0|    1138.0|   37.86|  -122.22|                   3.585|  6
.238137082601054|
|          352100.0|7.257399999999999|              52.0|     1467.0|
190.0|     496.0|     177.0|   37.85|  -122.24|                   3.521|  8.
288135593220339|
+------------------+-----------------+------------------+-----------+-
------------+----------+----------+--------+---------+--------------
---------+-------------------+
only showing top 3 rows
```

SELECT AND STANDARDIZE FEATURES (2 POINTS)

In [85]:
```
# retain these predictors for Part 1
vars_to_keep = ["median_house_value_final",
                "total_bedrooms",
                "population",
                "households",
                "median_income",
                "rooms_per_household"]

# subset the dataframe on these predictors
dfs = df.select(vars_to_keep)
dfs.show(3)
```

```
+------------------------+--------------+----------+----------+-------
----------+-------------------+
|median_house_value_final|total_bedrooms|population|households|    median_i
ncome|rooms_per_household|
+------------------------+--------------+----------+----------+-------
----------+-------------------+
|                   4.526|         129.0|     322.0|     126.0|           8.3
252|  6.984126984126984|
|                   3.585|        1106.0|    2401.0|    1138.0|           8.3
014|  6.238137082601054|
|                   3.521|         190.0|     496.0|     177.0|7.257399999999
999|  8.288135593220339|
+------------------------+--------------+----------+----------+-------
----------+-------------------+
only showing top 3 rows
```

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

In [86]:
```
from pyspark.ml.linalg import DenseVector
from pyspark.ml.feature import StandardScaler
input_data = dfs.rdd.map(lambda x: (x[0], DenseVector(x[1:])))
input_data.take(3)
```

Out[86]:
```
[(4.526, DenseVector([129.0, 322.0, 126.0, 8.3252, 6.9841])),
 (3.585, DenseVector([1106.0, 2401.0, 1138.0, 8.3014, 6.2381])),
 (3.521, DenseVector([190.0, 496.0, 177.0, 7.2574, 8.2881]))]
```

In [87]:
```
# create a dataframe
df2 = spark.createDataFrame(input_data, ['label','features'])
df2.show(3)
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|4.526|[129.0,322.0,126....|
|3.585|[1106.0,2401.0,11...|
|3.521|[190.0,496.0,177....|
+-----+--------------------+
only showing top 3 rows
```

In [88]:
```
# Feature scaling

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

In [89]:
```
# Fit the DataFrame to the scaler; this computes the mean, standard deviation of each feature
scaler = standardScaler.fit(df2)
```

In [90]:
```
# Transform the data in `df2` with the scaler
scaled_df = scaler.transform(df2)
```

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

In [91]:
```
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)**

In [92]:
```
# lecture note 12
from pyspark.ml.regression import LinearRegression

lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

Fit the model using the training data

In [93]:
```
lrModel = 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

In [69]:
```
# https://towardsdatascience.com/building-a-linear-regression-with-pyspark-and-mllib-d065c3ba246a
lrPred = lrModel.transform(test_data)
lrPred.show(5)
```

```
+-----+--------------------+--------------------+------------------+
|label|            features|     features_scaled|        prediction|
+-----+--------------------+--------------------+------------------+
|0.225|[73.0,216.0,63.0,...|[0.17329463000310...|1.7384319406190645|
| 0.25|[33.0,64.0,27.0,0...|[0.07833866835757...| 1.236167661520696|
|0.342|[153.0,112.0,47.0...|[0.36320655329418...|1.2940776636654883|
|0.379|[756.0,1798.0,749...|[1.79466767510069...|1.4828653758102242|
|  0.4|[185.0,318.0,115....|[0.43917132261062...|1.4644369543261706|
+-----+--------------------+--------------------+------------------+
only showing top 5 rows
```

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()`

In [71]:
```
# https://blog.epigno.systems/2018/02/18/machine-learning-with-pyspark-linear-regression/
from pyspark.ml.evaluation import RegressionEvaluator
eval = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")

# Mean Square Error
mse = eval.evaluate(lrPred, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)
```

```
MSE: 0.758
```

RESULTS SECTION

In [107]:
```
# Code for Part 2
# adding a new feature "housing_median_age" on top of Part 1

vars_to_keep_2 = ["median_house_value_final",
             "total_bedrooms",
             "population",
             "households",
             "median_income",
             "rooms_per_household",
             "housing_median_age"]
```

```
dfs_2 = df.select(vars_to_keep_2)

input_data_2 = dfs_2.rdd.map(lambda x: (x[0], DenseVector(x[1:])))

df2_2 = spark.createDataFrame(input_data_2, ['label','features'])

standardScaler_2 = StandardScaler(inputCol="features", outputCol="features
_scaled",
                                  withStd=True, withMean=False)

scaler_2 = standardScaler_2.fit(df2_2)
scaled_df_2 = scaler_2.transform(df2_2)

train_data_2, test_data_2 = scaled_df_2.randomSplit(train_test, seed)

lr_2 = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
lrModel_2 = lr_2.fit(train_data_2)

lrPred_2 = lrModel_2.transform(test_data_2)

mse_2 = eval.evaluate(lrPred_2, {eval.metricName: "mse"})
print("MSE: %.3f" % mse_2)
```

```
MSE: 0.758
```

In [108]:

```
# Code for Part 3
# adding an engineered feature "bedrooms_per_household" on top of Part 1

df = df.withColumn('bedrooms_per_household', df.total_bedrooms / df.househ
olds)

vars_to_keep_3 = ["median_house_value_final",
              "total_bedrooms",
              "population",
              "households",
              "median_income",
              "rooms_per_household",
              "bedrooms_per_household"]

dfs_3 = df.select(vars_to_keep_3)

input_data_3 = dfs_3.rdd.map(lambda x: (x[0], DenseVector(x[1:])))

df2_3 = spark.createDataFrame(input_data_3, ['label','features'])

standardScaler_3 = StandardScaler(inputCol="features", outputCol="features
_scaled",
                                  withStd=True, withMean=False)

scaler_3 = standardScaler_3.fit(df2_3)
scaled_df_3 = scaler_3.transform(df2_3)

train_data_3, test_data_3 = scaled_df_3.randomSplit(train_test, seed)

lr_3 = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
lrModel_3 = lr_3.fit(train_data_3)
```

```
lrPred_3 = lrModel_3.transform(test_data_3)

mse_3 = eval.evaluate(lrPred_3, {eval.metricName: "mse"})
print("MSE: %.3f" % mse_3)
```

```
MSE: 0.758
```

In [110]:

```
# Code for Part 4
# https://spark.apache.org/docs/2.2.0/ml-classification-regression.html
# "By setting [elasticNetParam] properly, elastic net contains both L1 and
L2 regularization as special cases.
# For example, if a linear regression model is trained with the [elasticNet
Param] set to 1, it is equivalent to a Lasso model."
lr_4 = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=1)
lrModel_4 = lr_4.fit(train_data)

lrPred_4 = lrModel_4.transform(test_data)

mse_4 = eval.evaluate(lrPred_4, {eval.metricName: "mse"})
print("MSE: %.3f" % mse_4)
```

```
MSE: 0.777
```

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

In [112]:

```
# print dataframe containing question_part, MSE

df_mse = spark.createDataFrame(
    [
        (1, mse), # create your data here, be consistent in the types.
        (2, mse_2),
        (3, mse_3),
        (4, mse_4),
    ],
    ['question_part', 'MSE'] # add your columns label here
)
df_mse.show()
```

```
+-------------+------------------+
|question_part|               MSE|
+-------------+------------------+
|            1|0.7581158452509782|
|            2|0.7581159623910975|
|            3|0.7581158671684333|
|            4|0.7774589810765221|
+-------------+------------------+
```