Konut Fiyatlarının Tahmini için Spark ile Regresyon Modeli Oluşturulması

Amaç ve Kapsam: Tanımlanan veri seti üzerinde konut fiyatlarının tahminlenmesi için Spark ML kütüphanesi kullanılarak PySpark ile bir regresyon modeli oluşturulacaktır.

Veri kümesi: California Housing Prices

https://www.kaggle.com/datasets/camnugent/california-housing-prices

Ortam: Proje kaggle'de bulunan notebook ortamında python ve temelinde pyspark kütüphanesi ile yapılmıştır.

BAŞLANGIÇ: & Veri Yükleme:

```
% pip install pyspark
```

- İlk önce notebook ortamımıza pyspark kütüphanesini yüklüyoruz.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
import pandas as pd
import numpy as np
```

- pyspark.sql ile dataframe yükleme işlemlerini yapabileceğiz. Pandas ve numpy hazırda işimiz olacağı kesin olduğu için onları da yükledik.

```
# Spark session başlatma
spark = SparkSession.builder \
    .appName("California Housing Prices Regression") \
    .getOrCreate()
# Veri setini yükleme
```

```
file_path = "/kaggle/input/california-housing-prices/housing.
housing_data = spark.read.csv(file_path, header=True, inferSc
```

- Burada spark variable ile spark session başlatıyoruz ve ardından verimizi spark.read.csv methodu ile housing_data isminde tutuyoruz.
- Spark variable output'u aşağıdadır →

```
SparkSession - in-memory

SparkContext

Spark UI

Versionv3.5.1
Masterlocal[*]
AppNameCalifornia Housing Prices Regression
```

Keşifşel Veri Analizi:

Ön Veri İnceleme:

- Spark'ta veri setini görmek için .show() methodu kullanılıyor.

```
housing_data.show(5)

+---------+--------+------------------+-----------+----------
|longitude|latitude|housing_median_age|total_rooms|total_bedr
+---------+--------+------------------+-----------+----------
|  -122.23|   37.88|              41.0|      880.0|         1
|  -122.22|   37.86|              21.0|     7099.0|        11
|  -122.24|   37.85|              52.0|     1467.0|         1
|  -122.25|   37.85|              52.0|     1274.0|         2
|  -122.25|   37.85|              52.0|     1627.0|         2
+---------+--------+------------------+-----------+----------
only showing top 5 rows
```

- Longitude: Her blok grubu için, bir coğrafi yerin kuzey veya güney yönündeki dünya ekvatorunun açısal mesafesi.

- Latitude : Her blok grubu için, bir coğrafi yerin doğu veya batı yönündeki dünya ekvatorunun açısal mesafesi.
- Housing Median Age: Bir blok grubuna ait insanların ortanca yaşıdır. Ortanca, gözlemlenen değerlerin frekans dağılımının orta noktasında bulunan değerdir.
- Total Rooms: Bir blok grubundaki evlerin toplam odalarının sayısı.
- Total Bedrooms: Bir blok grubundaki evlerin toplam yatak odalarının sayısı.
- Population: Bir blok grubunun nüfusu.
- Households: Bir blok grubundaki ev birimleri ve bu birimlerdeki sakinlerin sayısı.
- Median Income: Bir blok grubuna ait insanların ortanca gelirini kaydetmek için kullanılır.
- Median House Value: Bir blok grubuna ait ortanca ev değeridir (Hedef Değer).
- Öncelikle herhangi eksik değerleri kontrol edelim. →

```
# Eksik değerlerin kontrol edilmesi ve eksik değer içeren süt
missing_col = {}
for col_name in housing_data.columns:
    missing_count = housing_data.where(col(col_name).isNull()
    if missing_count > 0:
        missing_col[col_name] = missing_count

# Eksik değer içeren sütunların ve eksik değer sayılarının li
print("Eksik Değer İçeren Sütunlar ve Sayıları:")
for col_name, count in missing_col.items():
    print(col_name, ": ", count)
```

🔑 Output →

```
Eksik Değer İçeren Sütunlar ve Sayıları:
total_bedrooms :  207
```

Sadece total_bedrooms sütününda 207 satırda boş değerlerimiz vardır. Median değerlerine ya da bir ml model ile bu boş değerler doldurulabilir. Bu projede az sayıda boş değer olduğu için bu satırlar drop edilecektir.

- Describe ile veri özelliklerin değerlerine bakalım →

```
housing_data.describe().show()
```

🔑 Output:

```
+-------+------------------+-----------------+----
--------------+-----------------+----------------
-+-----------------+----------------+-----------
------+------------------+---------------+
|summary|         longitude|         latitude|hous
ing_median_age|       total_rooms|    total_bedroom
s|        population|       households|     median_
income|median_house_value|ocean_proximity|
+-------+------------------+-----------------+----
--------------+-----------------+----------------
-+------------------+----------------+-----------
------+------------------+---------------+
|  count|             20640|            20640|
20640|             20640|             20433|
20640|            20640|            20640|
20640|          20640|
|   mean|-119.56970445736148| 35.6318614341087|28.6
39486434108527|2635.7630813953488| 537.870552537561
8|1425.4767441860465|499.5396802325581|3.8706710029
070246|206855.81690891474|           NULL|
| stddev|  2.003531723502584|2.135952397457101| 12.
585557612111163|2181.6152515827944|421.3850700740311
5|  1132.46212176534|382.3297528316098| 1.899821717
945263|115395.61587441359|           NULL|
|    min|            -124.35|            32.54|
1.0|              2.0|              1.0|
3.0|             1.0|           0.4999|
14999.0|      <1H OCEAN|
|    max|            -114.31|            41.95|
52.0|           39320.0|            6445.0|
35682.0|           6082.0|           15.0001|
500001.0|     NEAR OCEAN|
+-------+------------------+-----------------+----
--------------+-----------------+----------------
```

```
-+------------------+-----------------+------------
------+------------------+---------------+
```

ocean_proximity sütünu hariç hepsi nümerik değer olduğu gözüküyor.

total_rooms, total_bedrooms, population, households, median_income, median_house_value değerlerinde min-max değerler arası çok açık. Standartize edilmesi gerek.

ocean_proximity encode edilecek.

longtitude, latitude değerleri id değerleri olduğu için gereksizdir. Droplanacaktır.

- Veri seti şeması →

```
housing_data.printSchema()
```

```
# ocean_proximity sütunundaki benzersiz değerlerin ve sayıları
unique_values_counts = housing_data.groupBy("ocean_proximity"

# Sonuçları görüntüleme
unique_values_counts.show()
```

Öznitelik Mühendisliği:

```
#id değerler droplanıyor.
housing_data = housing_data.drop("longitude", "latitude")
```

- Hedef değerimizi hesaplaması kolay olması amacıyla standarize etmeden önce daha düşük değerlere sıkıştırıyoruz.

```
# Adjust the values of `medianHouseValue`
housing_data = housing_data.withColumn("median_house_value",
```

```
splits = housing_data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1]
train_rows = train.count()
test_rows = test.count()
print("Training Rows:", train_rows, " Testing Rows:", test_ro
```

🔑 Verimizi data leak olmaması amacıyla şimdiden train ve test olarak bölüyoruz.

```
Training Rows: 14348  Testing Rows: 6292
```

```
# Tüm numerik sütunların seçilmesi
numeric_columns = [col for col, dtype in housing_data.dtypes
```

🔑 numeric_columns:

```
['housing_median_age',
 'total_rooms',
 'total_bedrooms',
 'population',
 'households',
 'median_income',
 'median_house_value']
```

```
# Get the columns and their data types
column_types = train.dtypes

# Identify categorical columns
categorical_columns = [col_name for col_name, col_type in col

# Print the categorical columns
print("Categorical Columns:")
```

```
for col_name in categorical_columns:
    print(col_name)
```

🔑 Categorical Columns:

```
ocean_proximity
```

Pipeline:

```
#gerekli kütüphaneler
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.sql.functions import col, round
```

- Numerik değerlerin virgülden sonrası fazla uzun olduğu için hesaplama kolaylığı amacıyla 2 ile yuvarlıyoruz.

```
# Eksik değerleri içeren satırların droplanması ve sayısal sü
train = train.na.drop()
for numeric_col in numeric_columns:
    train = train.withColumn(numeric_col, round(col(numeric_c
```

Pipeline'in bileşenlerini tanımlama.

```
#ilk aşama kategorik sütünları encode işlemi.
#encodelanan sütünlar farklı bir sütünda oluşacaktır.
#Spark'ın StringIndexer methodu ile yapıyoruz.
encoding = [StringIndexer(inputCol=col, outputCol=col+"_encod
```

```
# VectorAssembler kullanarak tüm sütunları birleştirme.
# Features isminde yeni sütünda matris olarak tutulacak tüm d
assembler_inputs = numeric_columns + [col + "_encoded" for co
assembler = VectorAssembler(inputCols=assembler_inputs, outpu
```

```
# StandardScaler ile sayısal sütunları ölçeklendirme.
# Yeni oluşturulan features sütününa yapılıyor.
scaler = StandardScaler(inputCol="features", outputCol="scale
```

```
# Pipeline oluşturma
pipeline = Pipeline(stages=encoding + [assembler, scaler])

# Pipeline'i kullanarak train veri setini dönüştürme
transformed_train = pipeline.fit(train).transform(train)

# Kategorik sütunları çıkar
transformed_train = transformed_train.drop(*categorical_colum

# İlk 5 satırı gösterme
transformed_train.show(5)
```

🔑 Transformed_train:

```
+------------------+-----------+--------------+----
------+----------+-------------+-----------------+
----------------------+--------------------+------
--------------+
|housing_median_age|total_rooms|total_bedrooms|popu
lation|households|median_income|median_house_value|
ocean_proximity_encoded|            features|     s
caled_features|
+------------------+-----------+--------------+----
------+----------+-------------+-----------------+
----------------------+--------------------+------
--------------+
|               1.0|       83.0|          15.0|
32.0|      15.0|         4.88|               1.42|
1.0|[1.0,83.0,15.0,32...|[-2.2050805988832...|
|               1.0|     2062.0|         343.0|
872.0|     268.0|          5.26|               1.91|
1.0|[1.0,2062.0,343.0...|[-2.2050805988832...|
|               2.0|      158.0|          43.0|
94.0|      57.0|         2.56|                0.6|
3.0|[2.0,158.0,43.0,9...|[-2.1254530465035...|
|               2.0|      200.0|          20.0|
25.0|       9.0|         15.0|                3.5|
0.0|[2.0,200.0,20.0,2...|[-2.1254530465035...|
|               2.0|      227.0|          35.0|
114.0|      49.0|         3.16|                4.35|
0.0|[2.0,227.0,35.0,1...|[-2.1254530465035...|
+------------------+-----------+--------------+----
------+----------+-------------+-----------------+
----------------------+--------------------+------
--------------+
only showing top 5 rows
```

Pipeline işlemini test veriseti için yapmak:

```
# Eksik değerleri içeren satırların droplanması ve sayısal sü
test = test.na.drop()
for numeric_col in numeric_columns:
    test = test.withColumn(numeric_col, round(col(numeric_col

# Pipeline'i kullanarak test veri setini dönüştürme
transformed_test = pipeline.fit(test).transform(test)

# Kategorik sütunları çıkar
transformed_test = transformed_test.drop(*categorical_columns

# İlk 5 satırı gösterme
transformed_test.show(5)
```

```
#scaled_features değerlerinde işlem yapacağımız için sütün is
cols = ["housing_median_age","total_rooms","total_bedrooms","
                        "median_house_value","ocean_proximity
```

ML Model

```
#gerekli kütüphaneler
from pyspark.ml.regression import LinearRegression
from pyspark.mllib.evaluation import RegressionMetrics

from pyspark.ml.tuning import ParamGridBuilder, CrossValidato
from pyspark.ml.feature import VectorAssembler, StandardScale
from pyspark.ml.evaluation import RegressionEvaluator
```

```
# `lr` model ataması
lr = (LinearRegression(featuresCol='scaled_features', labelCo
                               maxIter=10, regParam=0.3, elas
#model eğitimi transformed_train üzerinde scaled_features süt
#tahmin edilen değerler yeni sütünda pred isminde olacaktır.
#hedef değerimiz median_house_value olarak verilmiştir.
linearModel = lr.fit(transformed_train)
```

> “ElasticNet is a linear regression model trained with L1 and L2 prior as regularizer. This combination allows for learning a sparse model where few of the weights are non-zero like Lasso, while still maintaining the regularization properties of Ridge. We control the convex combination of L1 and L2 using the l1_ratio parameter.”

```
coeff_df = pd.DataFrame({"Feature": ["Intercept"] + cols, "Co
coeff_df = coeff_df[["Feature", "Co-efficients"]]
```

```
# Generate predictions
predictions = linearModel.transform(transformed_test)
# tahmin edilen değerler ile gerçek değerleri göster.
predandlabels = predictions.select("pred", "median_house_valu
predandlabels.show()
```

🔑 Sonuçlar şimdilik yakın gözüküyor.

```
+------------------+------------------+
|              pred|median_house_value|
+------------------+------------------+
|0.9330286783224904|              0.55|
|1.9480843818196418|              1.89|
|0.8800033803786094|              0.48|
| 3.735794426784775|              4.25|
| 2.781339063794916|              2.99|
|2.3344115525536324|               2.4|
|3.0237404258240863|              3.31|
| 2.008684722326935|              1.97|
| 1.440556530071066|              1.22|
|2.1753356587219894|              2.19|
| 2.069285062834227|              2.05|
|3.0161653832606747|               3.3|
|1.5617572110856515|              1.38|
| 1.607207466466121|              1.44|
|2.3722867653706907|              2.45|
|1.8874840413123495|              1.81|
| 3.925170490870064|               4.5|
|1.7587083177343525|              1.64|
|1.7890084879879988|              1.68|
|2.5919629997096263|              2.74|
+------------------+------------------+
only showing top 20 rows
```

- Sonuçların metrik değerlerine bakma vakti. Regresyon modeli olduğu için RMSE, MAE ve R2 değerlerine bakılacaktır.

```
# metrik değerleri tek tek gösterme
print("RMSE: {0}".format(linearModel.summary.rootMeanSquaredE
print("MAE: {0}".format(linearModel.summary.meanAbsoluteError
print("R2: {0}".format(linearModel.summary.r2))
```

```
RMSE: 0.2858953088118172
MAE: 0.22614458635472565
R2: 0.9388759811846737
```

- Spark'ın evaluator ile RMSE metriğine bakmak →

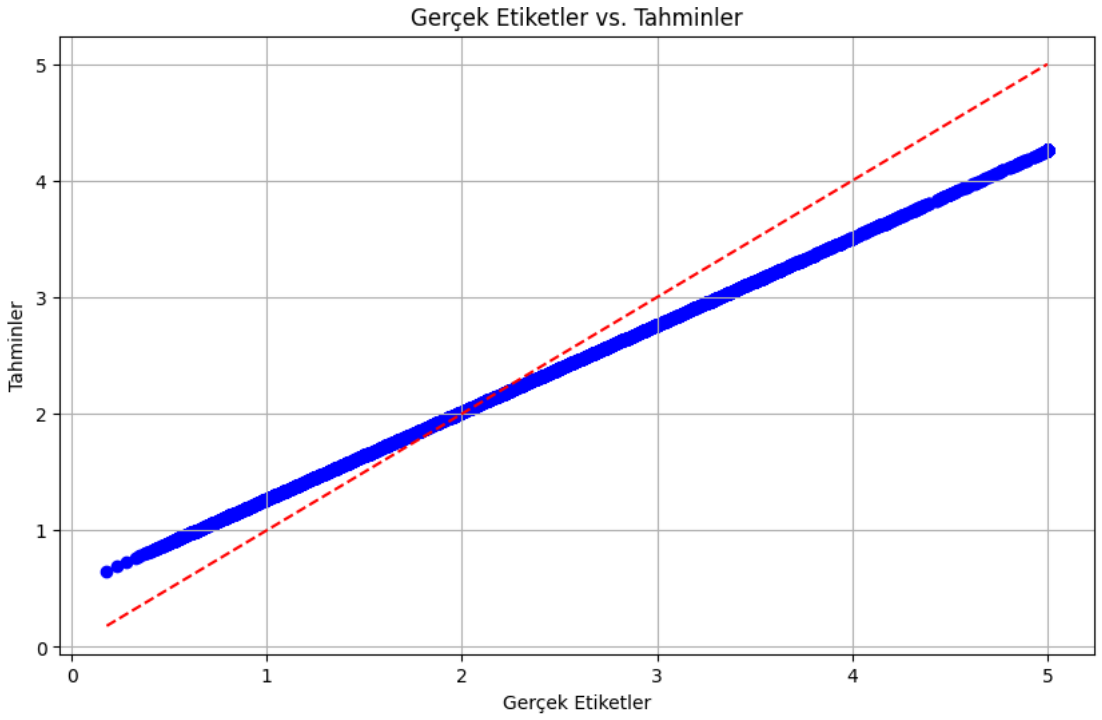
```
evaluator = RegressionEvaluator(predictionCol="pred", labelCo
print("RMSE: {0}".format(evaluator.evaluate(predandlabels)))
```

`RMSE: 0.2792060496240324`

Model Sonucu Grafiği:

```
import matplotlib.pyplot as plt

# Tahminler ve gerçek etiketlerle DataFrame oluşturma
predandlabels = predictions.select("pred", "median_house_valu

# RMSE değerini hesaplama
evaluator = RegressionEvaluator(predictionCol="pred", labelCo
rmse = evaluator.evaluate(predandlabels)

# Grafik çizimi için tahmin ve gerçek etiketleri bir listeye
pred_values = [row['pred'] for row in predandlabels.collect()
label_values = [row['median_house_value'] for row in predandl

# Grafik çizimi
plt.figure(figsize=(10, 6))
plt.scatter(label_values, pred_values, color='blue')
plt.plot([min(label_values), max(label_values)], [min(label_v
plt.xlabel('Gerçek Etiketler')
plt.ylabel('Tahminler')
plt.title('Gerçek Etiketler vs. Tahminler')
plt.grid(True)
plt.show()
```

```
# RMSE değerini yazdırma
print("RMSE:", rmse)
```




```
RMSE: 0.292285018825858
```

Son olarak Spark'ı durdurarak projeyi bitiriyoruz.

```
spark.stop()
```