predictive-maintenance-spark

June 14, 2023

[ ]:
```
import pandas as pd
from pyspark.sql.functions import corr
import numpy as np
import seaborn as sns
from pyspark.sql.functions import col
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml import Pipeline
import matplotlib.pyplot as plt
from pyspark.sql.functions import when
from pyspark.ml.classification import RandomForestClassifier
```

[ ]:
```
%fs
ls dbfs:/FileStore/tables/Rul/
```

[ ]:
```
#df=pd.read_csv('/dbfs/FileStore/tables/Rul/')
```

[ ]:
```
df = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/
 ↪tables/Rul/Battery_RUL.csv")
```

[ ]:
```
column_mapping = {
    "Cycle_Index": "Cycle_Index",
    "Discharge Time (s)": "Discharge_Time",
    "Decrement 3.6-3.4V (s)": "Decrement",
    "Max. Voltage Dischar. (V)": "Max_Voltage_Dischar",
    "Min. Voltage Charg. (V)": "Min_Voltage_Charg",
    "Time at 4.15V (s)": "Time_atV",
    "Time constant current (s)": "Time_constant_current",
    "Charging time (s)": "Charging_Time",
    "RUL": "New_RUL"
}
# Cambio dei nomi delle colonne
for old_name, new_name in column_mapping.items():
    df = df.withColumnRenamed(old_name, new_name)
```

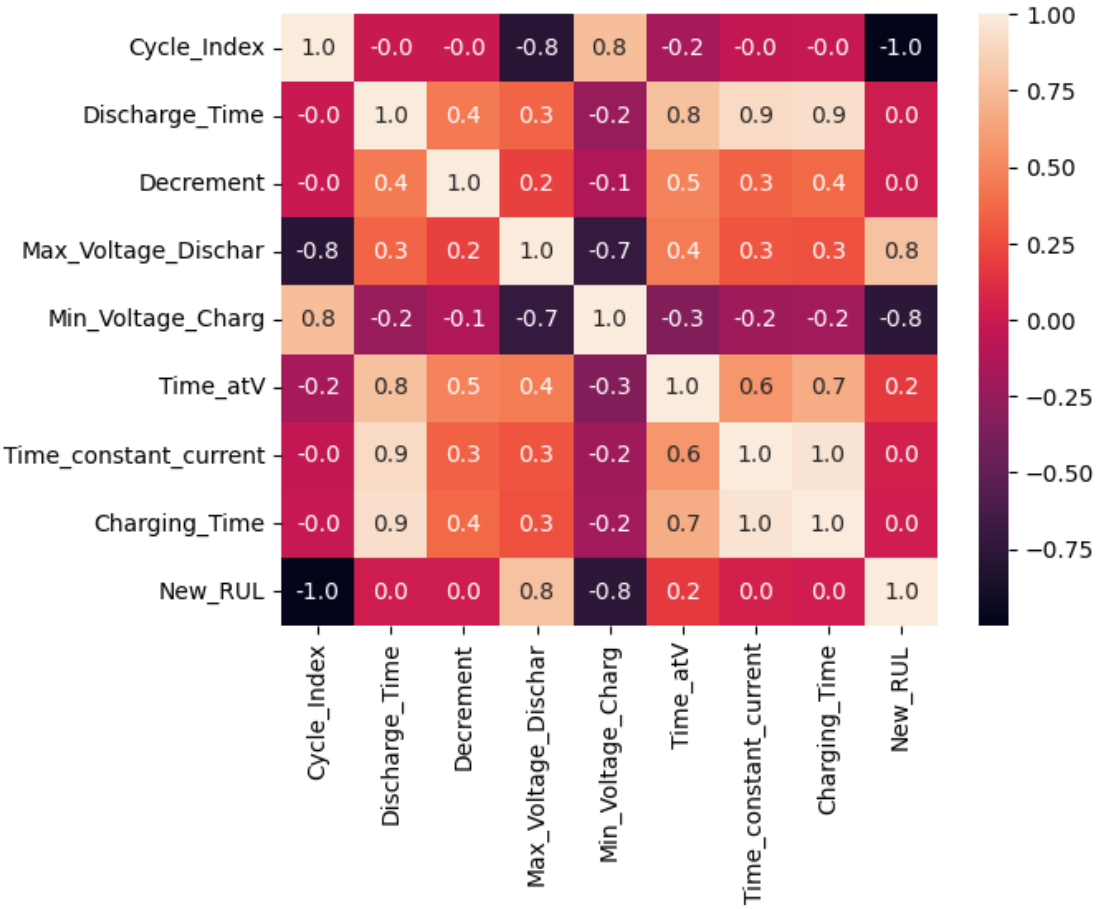
[ ]:
```
cor=[]
for col1 in df.columns:
```

```
    for col2 in df.columns:
        cor.append(df.select([corr(col1,col2)]).
 ↪alias(f"correlation_{col1}_{col2}").toPandas().values[0])
# Visualizza la matrice di correlazione
#print(correlation_matrix)
```

[ ]:
```
M_corr=pd.DataFrame(np.array(cor).reshape(9,9),columns=df.columns,index=df.
 ↪columns)
```

[ ]:
```
sns.heatmap(M_corr,annot=True, fmt=".1f")
```

[ ]: <AxesSubplot:>



[ ]:
```
columns=['Cycle_Index',
 'Max_Voltage_Dischar',
 'Min_Voltage_Charg',
 'Time_atV']
label='New_RUL'
```

[ ]:
```
columns_to_convert = columns

# Conversione delle colonne da stringa a float
for column in columns_to_convert:
    df = df.withColumn(column, col(column).cast("float"))
df = df.withColumn(label, col(label).cast("int"))
```

[ ]:
```
feature_columns =  ['Cycle_Index',
 'Max_Voltage_Dischar',
 'Min_Voltage_Charg',
 'Time_atV']
# Elenco delle colonne di input per il modello
label_column ='New_RUL' # Colonna di output da predire
# Crea un VectorAssembler per combinare le colonne delle features in un vettore
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

#data = assembler.transform(df1).select("features", label_column)
RFR = RandomForestRegressor(featuresCol="features",
                            labelCol=label_column,
                            )
pipeline = Pipeline(stages=[assembler,RFR])

# Addestra il modello sulla pipeline
model = pipeline.fit(df)
```

[ ]:
```
predictions = model.transform(df)
prediction1 =predictions.select('Cycle_Index',
 'Max_Voltage_Dischar',
 'Min_Voltage_Charg',
 'Time_atV',col("prediction").alias("p1"),when(predictions["prediction"] <␣
 ↪500,1).otherwise(0).alias("rul"))
```

[ ]:
```
display(prediction1)
```

[ ]:
```
train, test = prediction1.randomSplit([0.7, 0.3], seed=42)
```

[ ]:
```
feature_columns =  ['Cycle_Index',
 'Max_Voltage_Dischar',
 'Min_Voltage_Charg',
 'Time_atV',"p1"]
# Elenco delle colonne di input per il modello
label_column = "rul" # Colonna di output da predire
# Crea un VectorAssembler per combinare le colonne delle features in un vettore
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

#data = assembler.transform(df1).select("features", label_column)
```

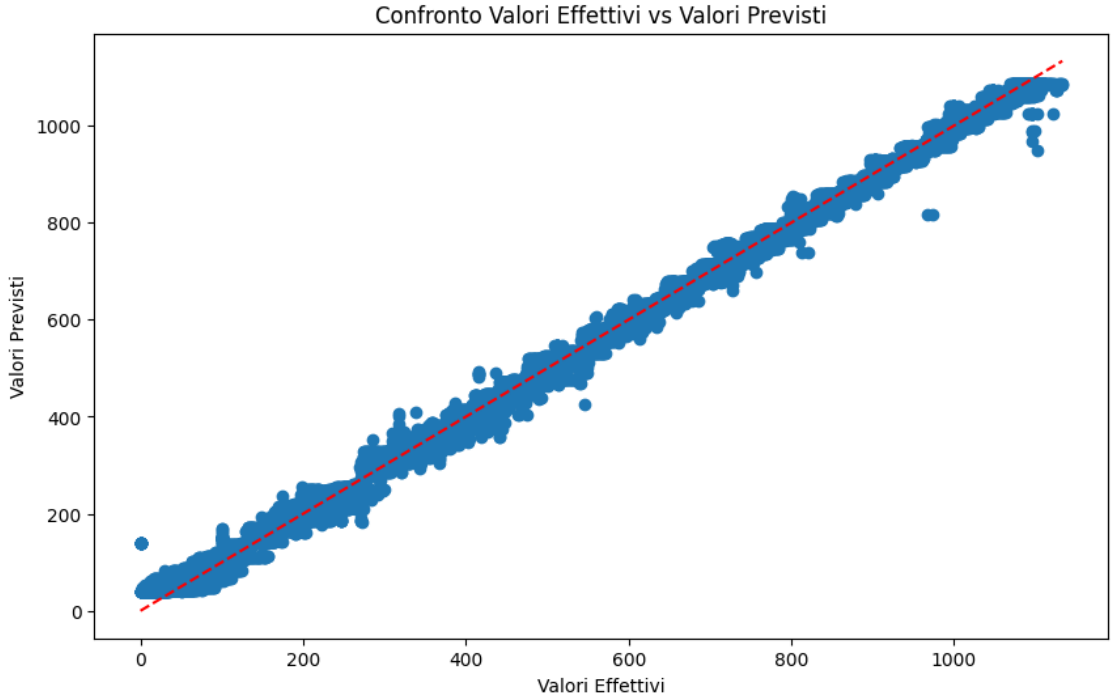
```
RFC = RandomForestClassifier(featuresCol="features",
                             labelCol=label_column,
                             )
pipeline = Pipeline(stages=[assembler,RFC])

# Addestra il modello sulla pipeline
model2= pipeline.fit(train)
```

[ ]:
```
prediction = model2.transform(train)
```

[ ]:
```
# Ottenere i valori previsti e i valori effettivi dal DataFrame delle predizioni
predicted_values = predictions.select("prediction").rdd.map(lambda row: row[0]).
 ↪collect()
actual_values = predictions.select('New_RUL').rdd.map(lambda row: row[0]).
 ↪collect()

# Creare il grafico di confronto
plt.figure(figsize=(10, 6))
plt.scatter(actual_values, predicted_values)
plt.plot([min(actual_values), max(actual_values)], [min(actual_values),␣
 ↪max(actual_values)], 'r--')
plt.xlabel("Valori Effettivi")
plt.ylabel("Valori Previsti")
plt.title("Confronto Valori Effettivi vs Valori Previsti")
plt.show()
```



[ ]:
```python
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Crea un oggetto BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol="rul")

# Calcola l'area sotto la curva ROC (AUC)
auc = evaluator.evaluate(prediction, {evaluator.metricName: "areaUnderROC"})
print("Area Under ROC Curve (AUC):", auc)
```

```
Area Under ROC Curve (AUC): 0.9999738285144567
```