Building a Machine Learning (ML) Model with PySpark

A step-by-step guide for beginners

Harun Ur Rashid Jun 20 · 5 min read ★

Design by myself

Spark is the name of the engine, that realizes cluster computing while PySpark is the Python's library to use Spark.

PySpark is a great language for performing exploratory data analysis at scale, building machine learning pipelines, and creating ETLs for a data platform. If you're already familiar with Python and libraries such as Pandas, then PySpark is a great language to learn in order to create more scalable analyses and pipelines.

The goal of this post is to show how to build an ml model using PySpark.

How To Install PySpark

PySpark installing process is very easy as like other python's packages. (eg. Pandas, Numpy,scikit-learn).

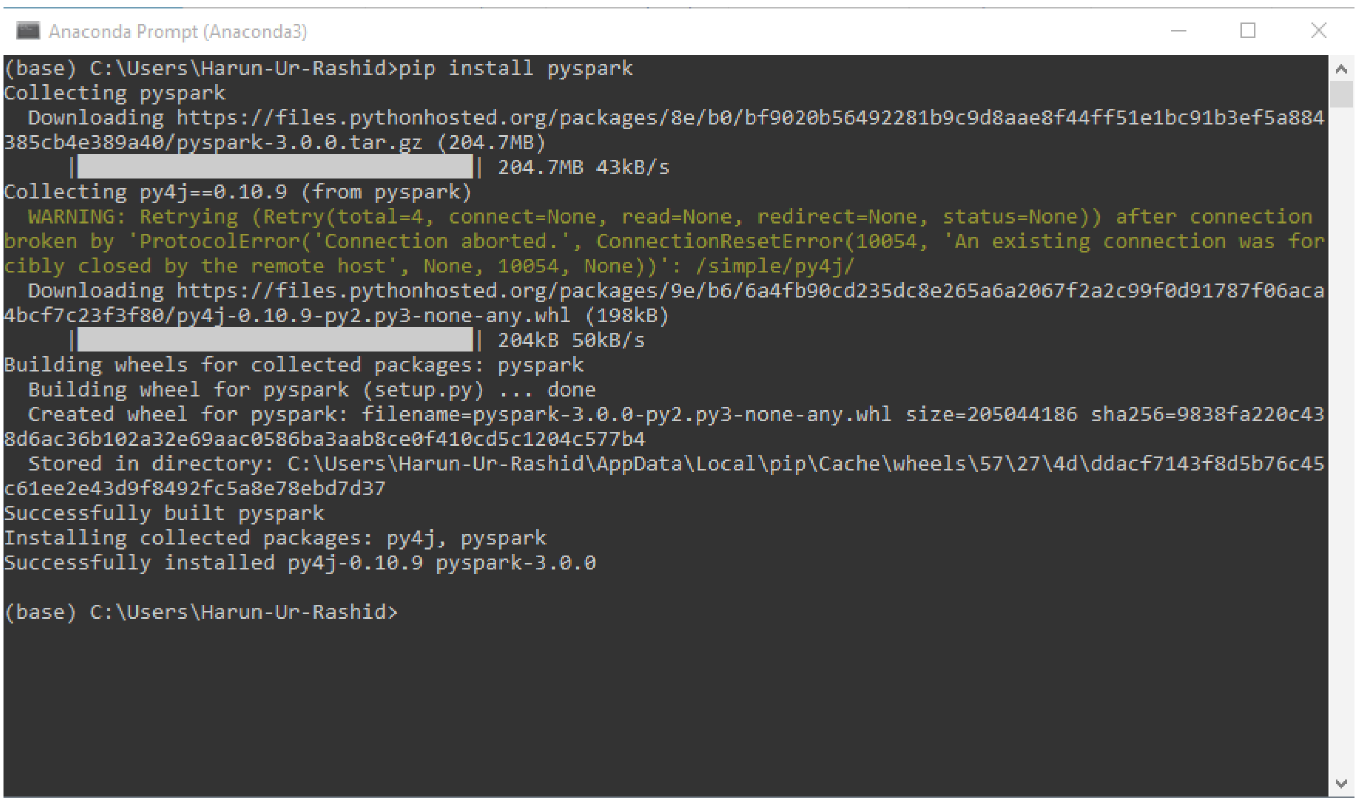
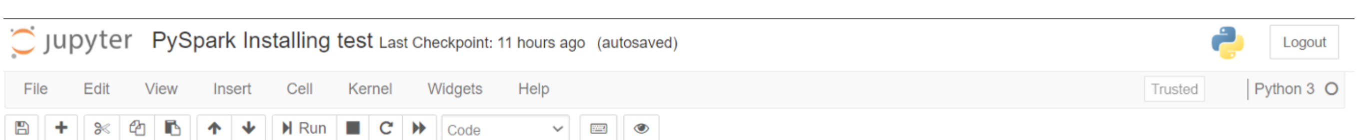


Figure 01

One important thing is to firstly ensure that java is installed to ensure java has installed in your machine. then you can run PySpark on your jupyter notebook.



```
In [1]: !pip install pyspark

        Requirement already satisfied: pyspark in c:\users\asus\anaconda3\lib\site-packages (3.0.0)
        Requirement already satisfied: py4j==0.10.9 in c:\users\asus\anaconda3\lib\site-packages (from pyspark) (0.10.9)
```

Done! ✌

```
In [2]: import pyspark

In [3]: pyspark.__version__
Out[3]: '3.0.0'
```

Congratulations!! 😃 ⚡

PySpark Checking in Jupyter Notebook

Exploring The Data

We will use the same data set when we built ml models in Python, and it is related to diabetes diseases of a National Institute of Diabetes and Digestive and Kidney Diseases. The classification goal is to predict whether the patient has diabetes (Yes/No). The dataset can be downloaded from Kaggle.

read_data.py

```
root
 |-- Pregnancies: integer (nullable = true)
 |-- Glucose: integer (nullable = true)
 |-- BloodPressure: integer (nullable = true)
 |-- SkinThickness: integer (nullable = true)
 |-- Insulin: integer (nullable = true)
 |-- BMI: double (nullable = true)
 |-- DiabetesPedigreeFunction: double (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Outcome: integer (nullable = true)
```

Figure 02

The datasets consist of several medical predictor variables and one target variable, Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

- Input Variables: Glucose,BloodPressure,BMI,Age,Pregnancies,Insulin,SkinThikness,DiabetesPedigreeFunction.

- Output variables: Outcome.

Have a peek of the first five observations. Pandas data frame is prettier than Spark DataFrame.show().

show_data.py

	0	1	2	3	4
Pregnancies	6.000	1.000	8.000	1.000	0.000
Glucose	148.000	85.000	183.000	89.000	137.000
BloodPressure	72.000	66.000	64.000	66.000	40.000
SkinThickness	35.000	29.000	0.000	23.000	35.000
Insulin	0.000	0.000	0.000	94.000	168.000
BMI	33.600	26.600	23.300	28.100	43.100
DiabetesPedigreeFunction	0.627	0.351	0.672	0.167	2.288
Age	50.000	31.000	32.000	21.000	33.000
Outcome	1.000	0.000	1.000	0.000	1.000

Figure 03

In PySpark you can show the data with Pandas' DataFrame using `toPandas()`

data_show_in_pandas_dataframe.py

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Figure 04

Checking the classes are perfectly balanced!!

class_balance_check.py

	Outcome	count
0	1	268
1	0	500

Figure 05

Statistics Summary

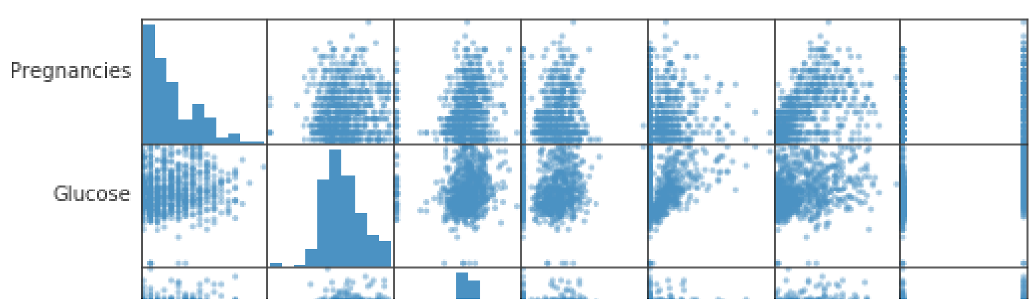
Statistics_Summary.py

	0	1	2	3	4
summary	count	mean	stddev	min	max
Pregnancies	768	3.8450520833333335	3.36957806269887	0	17
Glucose	768	120.89453125	31.97261819513622	0	199
BloodPressure	768	69.10546875	19.355807170644777	0	122
SkinThickness	768	20.536458333333332	15.952217567727642	0	99
Insulin	768	79.79947916666667	115.24400235133803	0	846
Age	768	33.240885416666664	11.760231540678689	21	81
Outcome	768	0.3489583333333333	0.476951377242799	0	1

Figure 06

Correlations between independent variables

correlations.py



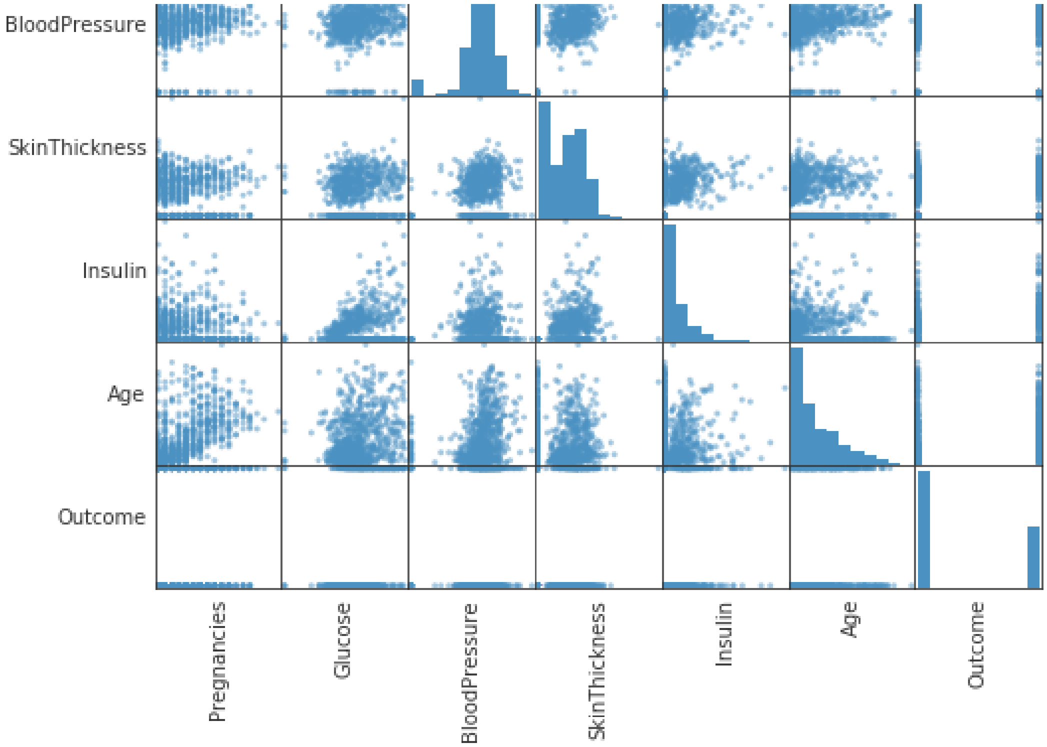


Figure 07

Data preparation and feature engineering

In this part, we will remove unnecessary columns and fill the missing values. Finally, we will select features for ml models. These features will be divided into two parts: train and test.

Let's starting the mission 🧑‍🚀

1. **Missing Data Handling:**

missing_data_handling.py

```
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+
|Pregnancies|Glucose|BloodPressure|SkinThickness|Insulin|BMI|DiabetesPedigreeFunction|Age|Outcome|
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+
|          0|      0|            0|            0|      0|  0|                       0|  0|      0|
+-----------+-------+-------------+-------------+-------+---+------------------------+---+-------+
```

Figure 08

Wow!! 👌 That's great in this datasets haven't any missing values.😀

2. Unnecessary columns dropping

columns_dropping.py

```
+-------+-------------+----+---+-------+
|Glucose|BloodPressure| BMI|Age|Outcome|
+-------+-------------+----+---+-------+
|    148|           72|33.6| 50|      1|
|     85|           66|26.6| 31|      0|
|    183|           64|23.3| 32|      1|
|     89|           66|28.1| 21|      0|
|    137|           40|43.1| 33|      1|
|    116|           74|25.6| 30|      0|
|     78|           50|31.0| 26|      1|
|    115|            0|35.3| 29|      0|
|    197|           70|30.5| 53|      1|
|    125|           96| 0.0| 54|      1|
|    110|           92|37.6| 30|      0|
|    168|           74|38.0| 34|      1|
|    139|           80|27.1| 57|      0|
|    189|           60|30.1| 59|      1|
|    166|           72|25.8| 51|      1|
|    100|            0|30.0| 32|      1|
|    118|           84|45.8| 31|      1|
|    107|           74|29.6| 31|      1|
|    103|           30|43.3| 33|      0|
|    115|           70|34.6| 32|      1|
+-------+-------------+----+---+-------+
only showing top 20 rows
```

Figure 09

3. Features Convert into Vector

VectorAssembler — a feature transformer that merges multiple columns into a vector column.

convert_to_vector.py

```
+-------+-------------+----+---+-------+--------------------+
|Glucose|BloodPressure| BMI|Age|Outcome|            features|
+-------+-------------+----+---+-------+--------------------+
|    148|           72|33.6| 50|      1|[148.0,72.0,33.6,...|
|     85|           66|26.6| 31|      0|[85.0,66.0,26.6,3...|
|    183|           64|23.3| 32|      1|[183.0,64.0,23.3,...|
|     89|           66|28.1| 21|      0|[89.0,66.0,28.1,2...|
|    137|           40|43.1| 33|      1|[137.0,40.0,43.1,...|
|    116|           74|25.6| 30|      0|[116.0,74.0,25.6,...|
|     78|           50|31.0| 26|      1|[78.0,50.0,31.0,2...|
|    115|            0|35.3| 29|      0|[115.0,0.0,35.3,2...|
|    197|           70|30.5| 53|      1|[197.0,70.0,30.5,...|
|    125|           96| 0.0| 54|      1|[125.0,96.0,0.0,5...|
|    110|           92|37.6| 30|      0|[110.0,92.0,37.6,...|
```

```
|    168|           74|38.0| 34|      1|[168.0,74.0,38.0,...|
|    139|           80|27.1| 57|      0|[139.0,80.0,27.1,...|
|    189|           60|30.1| 59|      1|[189.0,60.0,30.1,...|
|    166|           72|25.8| 51|      1|[166.0,72.0,25.8,...|
|    100|            0|30.0| 32|      1|[100.0,0.0,30.0,3...|
|    118|           84|45.8| 31|      1|[118.0,84.0,45.8,...|
|    107|           74|29.6| 31|      1|[107.0,74.0,29.6,...|
|    103|           30|43.3| 33|      0|[103.0,30.0,43.3,...|
|    115|           70|34.6| 32|      1|[115.0,70.0,34.6,...|
+-------+-------------+----+---+-------+--------------------+
only showing top 20 rows
```

Figure 10

Done!!✌️ Now features are converted into a vector.

4. Train and Test Split

Randomly split data into train and test sets, and set seed for reproducibility.

train_test_split.py

Training Dataset Count: 620

Test Dataset Count: 148

Machine learning Model Building

1. **Random Forest Classifier**

Random forest is a supervised learning algorithm which is used for both classification and regression cases, as well. But however, it is mainly used for classification problems. As we know that a forest is made up of trees and more trees mean more robust forests, in a similar way, random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

random_forest.py

Evaluate our Random Forest Classifier model.

random_forest_evaluate.py

Random Forest classifier Accuracy: 0.7945205479452054 (79.5%)

2. **Decision Tree Classifier**

Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, while not requiring feature scaling and are able to capture non-linearities and feature interactions.

decision_tree.py

```
+-------+-------------+----+---+-------+
|Glucose|BloodPressure| BMI|Age|Outcome|
+-------+-------------+----+---+-------+
|     57|           80|32.8| 41|      0|
|     67|           76|45.3| 46|      0|
|     71|           48|20.4| 22|      0|
|     71|           78|33.2| 21|      0|
|     72|           78|31.6| 38|      0|
|     76|           60|32.8| 41|      0|
|     78|           50|31.0| 26|      1|
|     78|           88|36.9| 21|      0|
|     84|           64|35.8| 21|      0|
|     84|           82|38.2| 23|      0|
+-------+-------------+----+---+-------+
only showing top 10 rows
```

Figure 11

Evaluate our Decision Tree model.

decision_tree_evaluate.py

Decision Tree Accuracy: 0.7876712328767124 (78.8%)

3. **Logistic Regression Model**

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. Logistic Regression is used when the dependent variable(target) is categorical.

logistic_regression.py

Evaluate our Logistic Regression model.

logistic_regression_evaluator.py

Logistic Regression Accuracy: 0.7876712328767124 (79.7%)

4. **Gradient-boosted Tree classifier Model**

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

g_boosted_tree.py

Evaluate our Gradient-Boosted Tree Classifier.

g_boosted_evalutor.py

Gradient-boosted Trees Accuracy: 0.8013698630136986(80.13%)

Conclusion

PySpark is a great language for data scientists to learn because it enables scalable analysis and ML pipelines. If you're already familiar with Python and Pandas, then much of your knowledge can be applied to Spark. To sum it up, we have learned how to build a machine learning application using PySpark. We tried three algorithms and gradient boosting performed best on our data set.

I got inspiration from Favio Vázquez's Github repository 'first_spark_model'.

Source code can be found on Github. I look forward to hearing feedback or questions.

> Machine learning models sparking when PySpark gave the accelerator gear like the need for speed gaming cars.

References:

1. Guru99, PySpark Tutorial for Beginners: Machine Learning Example

2. Apache Spark 3.0.0

3. Susan Li, Machine Learning with PySpark and MLlib — Solving a Binary Classification Problem

---

Thanks to Yenson Lau and Gerasimos Plegas.

Data Science Machine Learning Pyspark Python