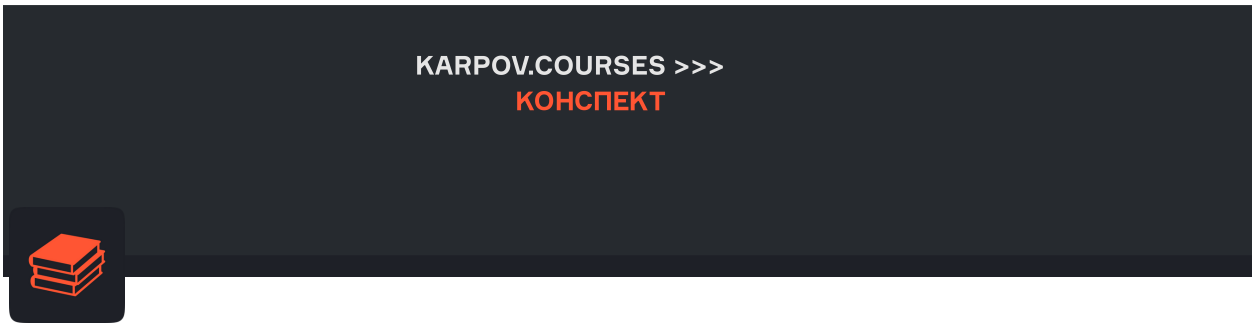

> Конспект > 5 урок > Практика: Применение не распределенных моделей МО на больших данных

> Оглавление

> Подготовка данных

В данной практике мы разберем как применять не распределенные модели МО на больших данных с использованием pyspark.

Будем работать с датасетом iris, в котором находятся данные о трех классах ирисов и их параметрах.

Будем решать задачу классификации, а именно определять сорт ириса на основе его параметров.

Импортируем нужные библиотеки.

```
import pickle    #  будем сохранять модель как pickle файл
import pandas as pd
```

```
import sklearn as sk
from sklearn import metrics
from sklearn.model_selection import train_test_split

from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.ml.feature import StringIndexer
from pyspark.sql.types import DoubleType
```

Создадим спарк сессию.

```
spark = SparkSession.builder.appName('PySparkTasks').getOrCreate()
```

Прочитаем датасет и посмотрим на имеющиеся параметры.

Видим, что в датасете присутствует всего 5 признаков. Целевым для нас является признак `species`, который описывает сорт ириса.

```
df = spark.read.parquet('iris.parquet')
df.show(5)

+------------+-----------+------------+-----------+-------+
|sepal_length|sepal_width|petal_length|petal_width|species|
+------------+-----------+------------+-----------+-------+
|         5.1|        3.5|         1.4|        0.2| setosa|
|         4.9|        3.0|         1.4|        0.2| setosa|
|         4.7|        3.2|         1.3|        0.2| setosa|
|         4.6|        3.1|         1.5|        0.2| setosa|
|         5.0|        3.6|         1.4|        0.2| setosa|
+------------+-----------+------------+-----------+-------+
```

`species` - это категориальный признак, чтобы его использовать нужно выполнить энкодинг, воспользуемся эстиматором `StringIndexer`.

Мы удаляем (метод `drop`) колонку `species`, вместо нее будем использовать `type`.

```
si = StringIndexer(inputCol='species', outputCol="type")
df = si.fit(df).transform(df).drop('species')

df.show(5)

+------------+-----------+------------+-----------+----+
|sepal_length|sepal_width|petal_length|petal_width|type|
+------------+-----------+------------+-----------+----+
```

```
|         5.1|        3.5|         1.4|        0.2| 0.0|
|         4.9|        3.0|         1.4|        0.2| 0.0|
|         4.7|        3.2|         1.3|        0.2| 0.0|
|         4.6|        3.1|         1.5|        0.2| 0.0|
|         5.0|        3.6|         1.4|        0.2| 0.0|
+------------+-----------+------------+-----------+----+
```

Преобразуем данные в pandas DataFrame.

```
pdf = df.toPandas()
pdf.head(10)
```

```
 +------------+-----------+------------+-----------+---+
 |sepal_length|sepal_width|petal_length|petal_width|type|
 +------------+-----------+------------+-----------+---+
0|         5.1|        3.5|         1.4|        0.2| 0.0|
1|         4.9|        3.0|         1.4|        0.2| 0.0|
2|         4.7|        3.2|         1.3|        0.2| 0.0|
3|         4.6|        3.1|         1.5|        0.2| 0.0|
4|         5.0|        3.6|         1.4|        0.2| 0.0|
5|         5.4|        3.9|         1.7|        0.4| 0.0|
6|         4.6|        3.4|         1.4|        0.3| 0.0|
7|         5.0|        3.4|         1.5|        0.2| 0.0|
8|         4.4|        2.9|         1.4|        0.2| 0.0|
9|         4.9|        3.1|         1.5|        0.1| 0.0|
 +------------+-----------+------------+-----------+----+
```

> Учим Sklearn модель

Полученный pandas DataFrame разделим на тренировочную и тестовую выборку.

```
train, test = train_test_split(pdf, random_state = 42)
```

Определяем список фичей, которые будут использованы для обучения. Признак `type` не включаем.

```
features_col = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
```

Сформируем тренировочные и тестовые выборки таким образом, чтобы туда входили только отобранные нами фичи. Так же зададим

переменные `y_train` и `y_test`, которые будут хранить целевой признак `type`.

```
X_train = train[features_col]
y_train = train.type    # y_train будет типа pandas.core.series.Series
X_test = test[features_col]
y_test = test.type
```

Для обучения выберем модель `DecisionTreeClassifier` из пакета `sklearn`. Создадим ее и обучим на тренировочном датасете.

```
from sklearn.tree import DecisionTreeClassifier


model = DecisionTreeClassifier(max_depth = 3, random_state = 1)
model = model.fit(X_train, y_train)
```

С помощью обученной модели выполним предсказания для тестовой выборки.

```
prediction=model.predict(X_test)
print(
    'The accuracy of the Decision Tree is {:.3f}'.format(
        metrics.accuracy_score(prediction, y_test)
    )
)

---
The accuracy of the Decision Tree is 1.000
```

Получили, что наша модель в 100% случаев правильно распознала сорт ирисов (такая большая точность получилась из-за небольшого размера исследуемого датасета).

Сохраним обученную модель.

```
with open('model.pickle', 'wb') as f:
    pickle.dump(model, f)
```

> Применяем Sklearn модель на больших данных

Применим нашу сохраненную модель в pyspark.

В SparkML любой алгоритм обучения модели является эстиматором, а любая обученная модель, которая применяется к данным, является трансформером.

Любая модель, которую мы обучили, будет некоторой функцией, которая может быть применена к имеющимся данным, чтобы по ним получить желаемый результат.

Поэтому для применения модели нам нужно создать pandas UDF. На вход такой UDF мы будем подавать список колонок, которые будут выступать фичами. На выходе мы получим pandas серию со значениями предсказаний (значения будут типа DoubleType).

```
import pyspark.sql.functions as F

@F.pandas_udf(returnType=DoubleType())
def predict_pandas_udf(*cols):
    import pandas as pd
    # cols will be a tuple of pandas.Series here.
    X = pd.concat(cols, axis=1) # собираем pd.DataFrame из переданных фичей
    with open('model.pickle', 'rb') as f:
        load_model = pickle.load(f)
    return pd.Series(load_model.predict(X))
```

Применим pandas UDF к нашему датафрему в pyspark. Создадим колонку `result`, куда будут записаны предсказания, полученные путем применения `predict_pandas_udf` к заданным `features_col`.

```
df_result = df.withColumn('result', predict_pandas_udf(*features_col))
```

Посмотрим, что получилось.

```
df_result.show()

+------------+-----------+------------+-----------+----+------+
|sepal_length|sepal_width|petal_length|petal_width|type|result|
```

```
+------------+-----------+------------+-----------+----+------+
|         5.1|        3.5|         1.4|        0.2| 0.0|   0.0|
|         4.9|        3.0|         1.4|        0.2| 0.0|   0.0|
|         4.7|        3.2|         1.3|        0.2| 0.0|   0.0|
...
```

Видим, что мы успешно применили нашу обученную не распределенную модель.

Оценим качество предсказаний модели.

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(
    labelCol="type", predictionCol="result", metricName="accuracy"
)
accuracy = evaluator.evaluate(df_result)
print("DecisionTreeClassifier [Accuracy] = %g"% (accuracy))
print("DecisionTreeClassifier [Error] = %g " % (1.0 - accuracy))
```

Получили, что

точность - DecisionTreeClassifier [Accuracy] = 0.966667

ошибка - DecisionTreeClassifier [Error] = 0.0333333

Точность отличается от того, что мы получали выше из-за того, что раньше мы применяли обученную модель только к тестовой выборке, а сейчас ко всему датафрейму.

Другой вариант применения не распределенных моделей - это встраивание их сразу в pipeline SparkML. Это может быть полезно, если нужно выполнить некие дополнительные преобразования. Для этого потребуется всего лишь реализовать трансформер, который будет оборачивать pandas UDF.