



> Конспект > 3 урок > Практика: Учет и трекинг моделей с ML Flow

> Оглавление

- > [Оглавление](#)
- > [Ручной трекинг эксперимента](#)
- > [Автотрекинг эксперимента](#)
- > [Взаимодействие с ML Flow](#)
 - > [Experiments](#)
 - > [Models](#)
 - > [Load model](#)

> Ручной трекинг эксперимента

Посмотрим, как можно производить трекинг эксперимента с помощью ML Flow.

Для работы нам понадобится библиотека pyspark.

Будем обучать модель на датасете по Титанику: модель будет учиться предсказывать, выживет или нет пассажир.

Загрузим нужные библиотеки, определим переменные окружения для доступа к S3 бакету YandexCloud, добавим ключи для доступа.

```
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, StringIndexerModel
from pyspark.ml.pipeline import PipelineModel
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Transformer

import mlflow
from mlflow.tracking import MlflowClient

import os
os.environ['MLFLOW_S3_ENDPOINT_URL'] = 'https://storage.yandexcloud.net'
os.environ['AWS_ACCESS_KEY_ID'] = '33kU43UzyCYfV1jgKUPL'
os.environ['AWS_SECRET_ACCESS_KEY'] = 'WPZnfkNE0lpdZ32hwVGhQ6PNiPPjmFZEajnwUMRe'
```

Для ML Flow законфигурируем хост.

Укажем эксперимент, который будем проводить и назовём его PySpark-ML.

```
mlflow.set_tracking_uri("https://mlflow.lab.karpov.courses")
mlflow.set_experiment(experiment_name = "PySpark-ML")
```

Запустим sparkсессию, проверим её. Она будет работать в локальном режиме.

```
spark = SparkSession.builder.appName("PySparkTitanikJob")\
    .getOrCreate()

spark
```

```
spark

SparkSession - in-memory

SparkContext

Spark UI

Version      v3.1.2
Master       local[*]
AppName      PySparkTitanikJob
```

Перед тем как начинать эксперимент, посмотрим на MLFlow.

The screenshot shows the MLFlow web interface. On the left, the 'Experiments' tab is active, displaying a list of experiments: 'PySpark-ML' and 'PySpark-ML-Auto'. The main area shows the details for the 'PySpark-ML' experiment (ID: 12). It includes a 'Notes' section, a 'Showing 9 matching runs' section with filters (Refresh, Compare, Delete, Download CSV, Start Time, All), and a table of runs. The table columns are Start Time, Duration, User, Source, Models, Metrics (accuracy, f1), and Parameters (stage_3_label, stage_3_type). The runs are sorted by start time, showing various durations and accuracy/f1 scores.

	Start Time	Duration	User	Source	Models	Metrics	Parameters
						accuracy	f1
<input type="checkbox"/>	4 minutes ago	28.1s	karpov	ipykernel_...	spark-mode.../24	0.824	0.821
<input type="checkbox"/>	12 hours ago	25.1s	karpov	ipykernel_...	spark-mode.../23	0.823	0.815
<input type="checkbox"/>	1 day ago	26.6s	karpov	ipykernel_...	spark-mode.../22	0.822	0.818
<input type="checkbox"/>	4 days ago	25.3s	karpov	ipykernel_...	spark-mode.../19	0.826	0.82
<input type="checkbox"/>	4 days ago	26.9s	karpov	ipykernel_...	spark-mode.../18	0.869	0.867
<input type="checkbox"/>	4 days ago	25.4s	karpov	ipykernel_...	spark-mode.../17	0.894	0.892
<input type="checkbox"/>	4 days ago	1.1min	karpov	ipykernel_...	spark-mode.../16	0.814	0.808
<input type="checkbox"/>	4 days ago	21.7min	karpov	ipykernel_...	-	0.843	0.837
<input type="checkbox"/>	4 days ago	4.6min	karpov	ipykernel_...	-	0.851	0.848

Слева видим названия экспериментов.

При выборе эксперимента увидим информацию о нём: как давно он проводился, сколько длился, кто проводил, какая модель была получена, залогированные метрики и параметры эксперимента.

Можем конфигурировать отображаемые колонки.

► Notes

Showing 9 matching runs

Refresh Compare Delete Download CSV Start Time

Columns Only show differences metrics.rmse < 1 and params.model = "tree" Search Filter Clear

Source	Models	accuracy	f1	Impurity	MaxDepth	stage_3_label
ipykernel_...	spark-mode.../24	0.824	0.821	0.0	5	Survived
ipykernel_...	spark-mode.../23	0.823	0.815	0.0	5	Survived
ipykernel_...	spark-mode.../22	0.822	0.818	0.0	5	Survived
ipykernel_...	spark-mode.../19	0.826	0.82	0.0	5	Survived
ipykernel_...	spark-mode.../18	0.869	0.867	0.0	5	Survived
ipykernel_...	spark-mode.../17	0.894	0.892	0.0	5	Survived
ipykernel_...	spark-mode.../16	0.814	0.808	0.0	5	Survived
ipykernel_...	-	0.843	0.837	0.0	5	Survived
ipykernel_...	-	0.851	0.848	0.0	5	Survived

Load more

Посмотрим более подробнее один из run-ов (запусков).

mlflow Experiments Models GitHub Docs

PySpark-ML > Run 0c0b11ef0d0548909c1f866776d50bea

Run 0c0b11ef0d0548909c1f866776d50bea

Date: 2021-11-21 13:55:40 Source: ipykernel_launcher.py User: karpov

Duration: 28.1s Status: FINISHED Lifecycle Stage: active

Notes

Parameters (15)

Metrics (2)

Tags

Artifacts

spark-model Full Path: s3://kc-mlflow/12/0c0b11ef0d0548909c1f866776d50bea/artifacts/spark-model spark-model, v24 Registered on 2021/11/21

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. This model is also registered to the [model registry](#).

Model schema

Input and output schema for your model. [Learn more](#)

Name	Type
No schema. See MLflow docs for how to include input and output schema with your model.	

Make Predictions

Predict on a Spark DataFrame:

```
import mlflow
logged_model = 'runs:/@c0b11ef0d0548909c1f866776d50bea/spark-model'

# Load model as a Spark UDF.
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model)

# Predict on a Spark DataFrame.
columns = list(df.columns)
df.withColumn('predictions', loaded_model(*columns)).collect()
```



























Predict on a Pandas DataFrame:

Видим когда он происходил, сколько длился и другие параметры, также хранится информация о моделях.

Запускаем наш run.

```
mlflow.start_run()
```

Он создаётся в ML Flow в нашем эксперименте.

						Metrics		Parameters >		
<input type="checkbox"/>	↓ Start Time	Duration	User	Source	Models	accuracy	f1	Impurity	MaxDepth	stage_3_label
<input type="checkbox"/>	4 seconds ago		karpov	 ipykernel_	-	-	-	-	-	-
<input type="checkbox"/>	 15 minutes ago	28.1s	karpov	 ipykernel_	 spark-mode.../24	0.824	0.821	0.0	5	Survived
<input type="checkbox"/>	 12 hours ago	25.1s	karpov	 ipykernel_	 spark-mode.../23	0.823	0.815	0.0	5	Survived
<input type="checkbox"/>	 1 day ago	26.6s	karpov	 ipykernel_	 spark-mode.../22	0.822	0.818	0.0	5	Survived
<input type="checkbox"/>	 4 days ago	25.3s	karpov	 ipykernel_	 spark-mode.../19	0.826	0.82	0.0	5	Survived
<input type="checkbox"/>	 4 days ago	26.9s	karpov	 ipykernel_	 spark-mode.../18	0.869	0.867	0.0	5	Survived
<input type="checkbox"/>	 4 days ago	25.4s	karpov	 ipykernel_	 spark-mode.../17	0.894	0.892	0.0	5	Survived
<input type="checkbox"/>	 4 days ago	1.1min	karpov	 ipykernel_	 spark-mode.../16	0.814	0.808	0.0	5	Survived
<input type="checkbox"/>	 4 days ago	21.7min	karpov	 ipykernel_	-	0.843	0.837	0.0	5	Survived
<input type="checkbox"/>	 4 days ago	4.6min	karpov	 ipykernel_	-	0.851	0.848	0.0	5	Survived
<div>Load more</div>										

На данный момент этот run пустой, т.к. мы ещё ничего в него не логировали и не завершили его.

Загружаем тренировочный DataFrame, разделяем датафрейм на тренировочный и тестовый.

```
titanic_df = spark.read.parquet('train.parquet')
titanic_df.show()
train, test = titanic_df.randomSplit([0.8, 0.2])
```

Создаём индексеры, чтобы заинкодить данные, создаём вектор ассемблер, указываем модель, собираем всё в пайплайн.

```
indexer_sex = StringIndexer(inputCol="Sex", outputCol="Sex_index")
indexer_embarked = StringIndexer(inputCol="Embarked", outputCol="Embarked_index")
feature = VectorAssembler(
    inputCols=["Pclass", "Age", "SibSp", "Parch", "Fare", "Family_Size", "Embarked_index", "Sex_index"],
    outputCol="features")
rf_classifier = RandomForestClassifier(labelCol="Survived", featuresCol="features")
pipeline = Pipeline(stages=[indexer_sex, indexer_embarked, feature, rf_classifier])
```

Учим пайплайн, применяем пайплайн на тестовых данных.

```
p_model = pipeline.fit(train)
prediction = p_model.transform(test)
```

Оцениваем качество модели.

```
evaluator_accuracy = MulticlassClassificationEvaluator(labelCol="Survived", predictionCol="prediction", metricName="accuracy")
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="Survived", predictionCol="prediction", metricName="f1")

p_accuracy = evaluator_accuracy.evaluate(prediction)
p_f1 = evaluator_f1.evaluate(prediction)
print(f"Pipeline model [Accuracy] = {p_accuracy}")
print(f"Pipeline model [F1] = {p_f1}")
```

Логлируем результаты нашей модели.

Пайплайн модель состоит из шагов (stage), и мы хотим логировать информацию по каждому stage-у. Логлируем его тип, для этого вызываем функцию log_param и передаём нужные параметры.

Далее логлируем разные stage-ы определённым образом. В данном случае у нас stage VectorAssembler и мы хотим залогировать список колонок, который превращается в feature вектор. Тоже самое делаем с string indexer, колонку, которую использует StringIndexer и какую создаст после инкодинга, далее stage-ы, которые могут быть моделью: у любой модели есть 2 параметра label – целевая переменная и колонка feature вектора.

```
for i in range(0, len(p_model.stages)):
    stage = p_model.stages[i]
    mlflow.log_param(f'stage_{i}_type', stage)
    if type(stage) is VectorAssembler:
```

```

mlflow.log_param(f'stage_{i}_input', stage.getInputCols())
mlflow.log_param(f'stage_{i}_output', stage.getOutputCol())
elif type(stage) is StringIndexerModel:
    mlflow.log_param(f'stage_{i}_input', stage.getInputCol())
    mlflow.log_param(f'stage_{i}_output', stage.getOutputCol())
else:
    mlflow.log_param(f'stage_{i}_features', stage.getFeaturesCol())
    mlflow.log_param(f'stage_{i}_label', stage.getLabelCol())

```

Если мы посмотрим параметры нашего run-а, то увидим что в нём залогировались параметры.

► Notes [🔗](#)

▼ Parameters (12)

Name	Value
stage_0_input	Sex
stage_0_output	Sex_index
stage_0_type	StringIndexerModel: uid=StringIndexer_f1ab9967c8d3, handleInvalid=error
stage_1_input	Embarked
stage_1_output	Embarked_index
stage_1_type	StringIndexerModel: uid=StringIndexer_9f893021219e, handleInvalid=error
stage_2_input	['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Family_Size', 'Embarked_index', 'Sex_index']
stage_2_output	features
stage_2_type	VectorAssembler_b8b6314f087d
stage_3_features	features
stage_3_label	Survived
stage_3_type	RandomForestClassificationModel: uid=RandomForestClassifier_48f5673b155e, numTrees=20, numClasses=2, numFeatures=8

Видим информацию о каждом stage-е.

Далее логируем данные по модели.

Можем взять из последнего stage-а параметры модели, которую мы обучили.

```

mlflow.log_param('MaxDepth', p_model.stages[-1].getMaxDepth())
mlflow.log_param('MaxNumTrees', p_model.stages[-1].getMaxBins())
mlflow.log_param('Impurity', p_model.stages[-1].getMinInfoGain())

```

Наш эксперимент ещё не завершён, он ждёт команды завершения.

Логируем измеренные метрики:

```

mlflow.log_metric('accuracy', p_accuracy)
mlflow.log_metric('f1', p_f1)

```

▼ Metrics (2)

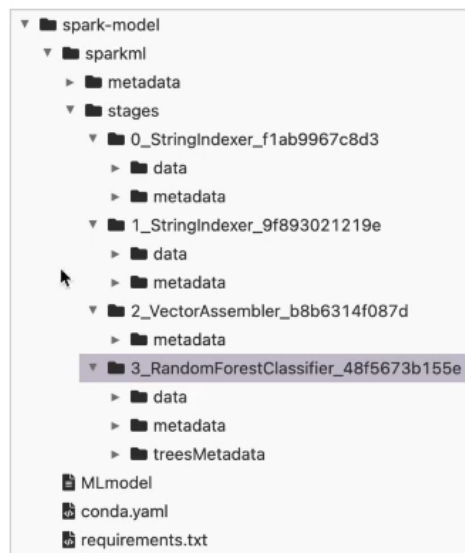
Name	Value
accuracy	0.826
f1	0.824

Далее залогируем саму обученную модель.

```
mv = mlflow.spark.log_model(p_model,
                             artifact_path = "spark-model",
                             registered_model_name="spark-model")
mv
```

Теперь видим набор артефактов, sparkml модель и все stage-ы/

▼ Artifacts



Есть отличие между spark и другими моделями: другие модели имеют pickle file в ML Flow, но для spark-а ML Flow сохраняет всё в виде набора папок.

Далее в MLmodel видим файл, который описывает нашу модель.

▼ Artifacts



В conda.yaml описана конфигурация фреймворков, которая необходима, чтобы модель воспроизводилась.

▼ Artifacts

spark-model

- sparkml
- MLmodel
- conda.yaml
- requirements.txt

Full Path: s3://kc-mlflow/12/79d1b47e4ea147ac93c2ecdbc0660888/artifacts/spark-model/conda.yaml
 Size: 114B

```

channels:
- conda-forge
dependencies:
- python=3.8.10
- pip
- pip:
- mlflow
- pyspark==3.1.2
name: mlflow-env
  
```

В requirements.txt указано что нужно установить.

▼ Artifacts

spark-model

- sparkml
- MLmodel
- conda.yaml
- requirements.txt

Full Path: s3://kc-mlflow/12/79d1b47e4ea147ac93c2ecdbc0660888/artifacts/spark-model/requirements.txt
 Size: 21B

```

mlflow
pyspark==3.1.2
  
```

Для завершения эксперимента вызываем:

```
mlflow.end_run()
```

Эксперимент завершился по умолчанию успешно.

The screenshot shows the MLflow web interface. At the top, there are tabs for 'Experiments' and 'Models'. Below the header, the experiment name 'PySpark-ML > Run 79d1b47e4ea147ac93c2ecdbc0660888' is displayed. The run ID 'Run 79d1b47e4ea147ac93c2ecdbc0660888' is also shown. The run details include: Date: 2021-11-21 14:10:56, Source: ipynote_launcher.py, User: karpov, Duration: 8.9min, Status: FINISHED, and Lifecycle Stage: active. Below the run details, there are sections for 'Notes', 'Parameters (15)', 'Metrics (2)', 'Tags', and 'Artifacts'. The 'Artifacts' section is expanded, showing a tree view of the artifacts: spark-model, sparkml, MLmodel, conda.yaml, and requirements.txt. The 'MLflow Model' section is also expanded, showing the model schema and the code snippets for making predictions using the logged model. The model schema is empty, and the code snippets show how to load the model as a Spark UDF and predict on a Spark DataFrame.

По эксперименту получена модель, которая имеет версию 25.

Видим подсказки о том, как мы можем применять модель.

Если мы кликнем на «spark-model, v25», то перейдём в раздел Models.

mlflow Experiments Models

Registered Models > spark-model > Version 25

Version 25

Registered At: 2021-11-21 14:17:34 Creator: Stage: None

Last Modified: 2021-11-21 14:17:34 Source Run: Run 79d1b47e4ea147ac93c2ecdbc0660888

▸ Description [Edit](#)

▸ Tags

▾ Schema

Name	Type
No schema. See MLflow docs for how to include input and output schema with your model.	

Также можем вернуться обратно на эксперимент кликнув на Source Run.

Сам раздел Models выглядит следующим образом:

mlflow Experiments Models GitHub Docs

Registered Models

Share and manage machine learning models. [Learn more](#)

Create Model

Search by model name Search Filter Clear

Name	Latest Version	Staging	Production	Last Modified	Tags
spark-model	Version 25	-	Version 22	2021-11-21 14:17:34	-

< Page 1 > 10 / page

Видим один тип зарегистрированных моделей spark-model, кликнув по нему увидим все версии моделей.

spark-model

Created Time: 2021-11-17 02:53:49 Last Modified: 2021-11-21 14:17:34

▸ Description [Edit](#)

▸ Tags

▾ Versions All Active 1 Compare

<input type="checkbox"/>	Version	Registered at	Created by	Stage	Description
<input type="checkbox"/>	Version 25	2021-11-21 14:17:34		None	
<input type="checkbox"/>	Version 24	2021-11-21 13:56:08		None	
<input type="checkbox"/>	Version 23	2021-11-21 01:33:07		None	
<input type="checkbox"/>	Version 22	2021-11-20 01:05:15		Production	
<input type="checkbox"/>	Version 21	2021-11-19 04:11:59		None	
<input type="checkbox"/>	Version 20	2021-11-19 04:09:05		None	
<input type="checkbox"/>	Version 19	2021-11-17 05:36:03		None	
<input type="checkbox"/>	Version 18	2021-11-17 05:26:44		None	
<input type="checkbox"/>	Version 17	2021-11-17 05:04:07		Archived	
<input type="checkbox"/>	Version 16	2021-11-17 04:37:17		None	

Видим, что одна модель назначена на Production, нашу модель v25 можем также назначить на Production, но обычно это делается через Staging.

mlflow

ExperimentsModels

Registered Models > spark-model > Version 25

Version 25

Registered At: 2021-11-21 14:17:34

Creator:

Stage: None

Last Modified: 2021-11-21 14:17:34

Source Run: Run 79d1b47e4ea147ac93c2ecdbc0660888

Description Edit

Tags

Schema

Name	Type
No schema. See MLflow docs for how to include input and output schema with your model.	

Transition to → Staging

Transition to → Production

Transition to → Archived

При назначении модели Production, предыдущая Production модель переходит в архив, т.к. Production модель может быть только одна.

spark-model

Created Time: 2021-11-17 02:53:49Last Modified: 2021-11-21 14:22:40

Description Edit

Tags

Versions

AllActive 1Compare

<input type="checkbox"/>	Version	Registered at	Created by	Stage	Description
<input type="checkbox"/>	Version 25	2021-11-21 14:17:34		Production	
<input type="checkbox"/>	Version 24	2021-11-21 13:56:08		None	
<input type="checkbox"/>	Version 23	2021-11-21 01:33:07		None	
<input type="checkbox"/>	Version 22	2021-11-20 01:05:15		Archived	
<input type="checkbox"/>	Version 21	2021-11-19 04:11:59		None	
<input type="checkbox"/>	Version 20	2021-11-19 04:09:05		None	
<input type="checkbox"/>	Version 19	2021-11-17 05:36:03		None	
<input type="checkbox"/>	Version 18	2021-11-17 05:26:44		None	
<input type="checkbox"/>	Version 17	2021-11-17 05:04:07		Archived	
<input type="checkbox"/>	Version 16	2021-11-17 04:37:17		None	

Мы провели эксперимент, получили модель, залогировали параметры и метрики. В эксперименте не фиксировался тестовый датасет, поэтому метрики будут немного отличаться между экспериментами.

> Автотрекинг эксперимента

В модуле ML Flow есть возможность автотрекинга, которая избавляет от написания дополнительного кода, но даёт меньше гибкости.

Подключаем те же библиотеки и устанавливаем такую же конфигурацию.

```
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer
from pyspark.ml.pipeline import PipelineModel
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

import mlflow

import os
```

```
os.environ['MLFLOW_S3_ENDPOINT_URL'] = 'https://storage.yandexcloud.net'
os.environ['AWS_ACCESS_KEY_ID'] = '33kU43UzyCYfV1jgKUPL'
os.environ['AWS_SECRET_ACCESS_KEY'] = 'WPZnfkNE0LpdZ32hwVGhQ6PNiPPjmFZEajnwUMRe'

mlflow.set_tracking_uri("https://mlflow.lab.karpov.courses")
mlflow.set_experiment(experiment_name = "PySpark-ML-Auto")
```

Создаём spark сессию с помощью пакета mlflow-spark, чтобы ML Flow смог интегрироваться в spark сессию.

```
spark = SparkSession.builder.appName("PySparkTitanikJob")\
    .config("spark.jars.packages", "org.mlflow:mlflow-spark:1.11.0")\
    .getOrCreate()
```

Запускаем автотрекинг.

```
mlflow.pyspark.ml.autolog()
```

Проведём аналогичную подготовку перед обучением модели и запустим обучение.

```
titanic_df = spark.read.parquet('train.parquet')
titanic_df.show()
train, test = titanic_df.randomSplit([0.8, 0.2])
indexer_sex = StringIndexer(inputCol="Sex", outputCol="Sex_index")
indexer_embarked = StringIndexer(inputCol="Embarked", outputCol="Embarked_index")
feature = VectorAssembler(
    inputCols=["Pclass", "Age", "SibSp", "Parch", "Fare", "Family_Size", "Embarked_index", "Sex_index"],
    outputCol="features")
rf_classifier = RandomForestClassifier(labelCol="Survived", featuresCol="features")
pipeline = Pipeline(stages=[indexer_sex, indexer_embarked, feature, rf_classifier])
p_model = pipeline.fit(train)
```

Когда вызвали `p_model = pipeline.fit(train)`, видим, что включился автотрекинг.

```
2021/11/21 11:34:38 INFO mlflow.utils.autologging_utils: Created MLflow autologging run with ID '9376af1edebf44c1bd6f1372e0c7a276', which will track hyperparameters, performance metrics, model artifacts, and lineage information for the current pyspark.ml workflow
```

Вызываем предсказание на тестовых данных.

```
prediction = p_model.transform(test)
```

Получаем метрики.

```
evaluator_accuracy = MulticlassClassificationEvaluator(labelCol="Survived", predictionCol="prediction", metricName="accuracy")
evaluator_f1 = MulticlassClassificationEvaluator(labelCol="Survived", predictionCol="prediction", metricName="f1")

p_accuracy = evaluator_accuracy.evaluate(prediction)
p_f1 = evaluator_f1.evaluate(prediction)
print(f"Pipeline model [Accuracy] = {p_accuracy}")
print(f"Pipeline model [F1] = {p_f1}")
```

Мы завершили процесс обучения модели.

Если мы обновим в ML Flow наш Auto эксперимент, то увидим, что добавилась ещё одна модель.

Experiments

Search Experiments

PySpark-ML

PySpark-ML-Auto

PySpark-ML-Auto

Track machine learning training runs in an experiment. [Learn more](#)

Experiment ID: 15

Notes

Showing 9 matching runs

Refresh Compare Delete Download CSV Start Time

Columns Only show differences metrics.rmse < 1 and params.model = "tree" Search Filter Clear

Start Time	Duration	Run Name	User	Source	Version	Models	Parameters	Tags
44 seconds ago	20.8s	-	karpov	ipykernel_	-	spark	True False 10	Pipeline
8 minutes ago	22.6s	-	karpov	ipykernel_	-	spark	True False 10	Pipeline
40 minutes ago	22.3s	-	karpov	ipykernel_	-	spark	True False 10	Pipeline
4 days ago	19.4s	-	karpov	ipykernel_	-	spark	True False 10	Pipeline
4 days ago	19.1s	-	karpov	ipykernel_	-	spark	True False 10	Pipeline
4 days ago	19.0s	-	karpov	ipykernel_	-	spark	True False 10	Pipeline
4 days ago	19.6s	-	karpov	ipykernel_	-	spark	True False 10	Pipeline
4 days ago	13.4s	-	karpov	ipykernel_	-	spark	True False 10	Pipeline
4 days ago	14.2s	-	karpov	ipykernel_	-	-	True False 10	Pipeline

Это и есть наш эксперимент, который автоматически залогировался в нашем run-e.

PySpark-ML-Auto > Run 9376af1edebf44c1bd6f1372e0c7a276

Run 9376af1edebf44c1bd6f1372e0c7a276

Date: 2021-11-21 14:34:38 Source: ipykernel_launcher.py User: karpov

Duration: 20.8s Status: FINISHED Lifecycle Stage: active

Notes

Parameters (32)

Metrics

Tags (2)

Artifacts

model Full Path: s3://kc-mlflow/15/9376af1edebf44c1bd6f1372e0c7a276/artifacts/model Register Model

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also [register it to the model registry](#) to version control

Model schema

Input and output schema for your model. [Learn more](#)

Name	Type
No schema. See MLflow docs for how to include input and output schema with your model.	

Make Predictions

Predict on a Spark DataFrame:

```
import mlflow
logged_model = 'runs:/9376af1edebf44c1bd6f1372e0c7a276/model'

# Load model as a Spark UDF.
loaded_model = mlflow.pyfunc.spark_udf(spark, model_uri=logged_model)

# Predict on a Spark DataFrame.
columns = list(df.columns)
df.withColumn('predictions', loaded_model(*columns)).collect()
```

Модель не зарегистрировалась на автомате, но её можно зарегистрировать, нажав на соответствующую кнопку.

В Parameters можно посмотреть набор параметров, которые залогировал автотрекинг.

Автотрекинг является экспериментальной функцией, чаще используется ручной трекинг, что позволяет более гибко логировать информацию о моделях.

> Взаимодействие с ML Flow

У нас в ML Flow есть несколько экспериментов и по ним имеется по несколько запусков, в ходе которых были получены разные модели.

Будем взаимодействовать с самим ML Flow.

Сконфигурируем начальные параметры, чтобы мы могли взаимодействовать с S3 бакетом.

```
import os
import datetime
os.environ['MLFLOW_S3_ENDPOINT_URL'] = 'https://storage.yandexcloud.net'
os.environ['AWS_ACCESS_KEY_ID'] = '33ku43UzyCYfV1jgKUPL'
os.environ['AWS_SECRET_ACCESS_KEY'] = 'WPZnfkNE0lpdZ32hwVGhQ6PNiPPjmFZEajnwUMRe'
```

Подключаем ML Flow пакет и ML Flow Client.

```
import mlflow
from mlflow.tracking import MlflowClient

mlflow.set_tracking_uri("https://mlflow.lab.karpov.courses")
client = MlflowClient()
```

> Experiments

Мы можем посмотреть список экспериментов, которые зарегистрированы в ML Flow.

```
client.list_experiments()
```

```
client.list_experiments()

[<Experiment: artifact_location='s3://kc-mlflow/12', experiment_id='12', lifecycle_stage='active', name='PySpark-ML', tags={}>,
 <Experiment: artifact_location='s3://kc-mlflow/15', experiment_id='15', lifecycle_stage='active', name='PySpark-ML-Auto', tags={}>,
 <Experiment: artifact_location='s3://kc-mlflow/19', experiment_id='19', lifecycle_stage='active', name='DriverClassification', tags={}>,
 <Experiment: artifact_location='s3://kc-mlflow/20', experiment_id='20', lifecycle_stage='active', name='a.savchenko', tags={}>]
```

Найдём эксперимент по имени и получим о нём информацию.

```
exp = client.get_experiment_by_name('PySpark-ML')
exp
```

```
exp = client.get_experiment_by_name('PySpark-ML')
exp

<Experiment: artifact_location='s3://kc-mlflow/12', experiment_id='12', lifecycle_stage='active', name='PySpark-ML', tags={}>
```

Теперь мы можем запросить у ML Flow список запусков у данного эксперимента.

```
client.list_run_infos(exp.experiment_id)
```

```
client.list_run_infos(exp.experiment_id)|
```

```
[<RunInfo: artifact_uri='s3://kc-mlflow/12/1ce9927f22a14719a2783875e4a159a6/artifacts', end_time=1637596278801, experiment_id='12', lifecycle_stage='active', run_id='1ce9927f22a14719a2783875e4a159a6', run_uuid='1ce9927f22a14719a2783875e4a159a6', start_time=1637596252449, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/79d1b47e4ea147ac93c2ecd06060888/artifacts', end_time=1637493591304, experiment_id='12', lifecycle_stage='active', run_id='79d1b47e4ea147ac93c2ecd06060888', run_uuid='79d1b47e4ea147ac93c2ecd06060888', start_time=1637493056487, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/0c0b11ef0d0548909c1f866776d50bea/artifacts', end_time=1637492169054, experiment_id='12', lifecycle_stage='active', run_id='0c0b11ef0d0548909c1f866776d50bea', run_uuid='0c0b11ef0d0548909c1f866776d50bea', start_time=1637492140948, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/6c0c2e3da3aa4c0e87198a4726e9002a/artifacts', end_time=1637447587351, experiment_id='12', lifecycle_stage='active', run_id='6c0c2e3da3aa4c0e87198a4726e9002a', run_uuid='6c0c2e3da3aa4c0e87198a4726e9002a', start_time=1637447562254, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/109b758056cb48b5b9ad92d0d70e3566/artifacts', end_time=1637359515339, experiment_id='12', lifecycle_stage='active', run_id='109b758056cb48b5b9ad92d0d70e3566', run_uuid='109b758056cb48b5b9ad92d0d70e3566', start_time=1637359488720, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/ba5eeb76a4544305bb0686e524acf443/artifacts', end_time=1637116563230, experiment_id='12', lifecycle_stage='active', run_id='ba5eeb76a4544305bb0686e524acf443', run_uuid='ba5eeb76a4544305bb0686e524acf443', start_time=1637116537975, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/3f1a65e9d6da477c892609764492217f/artifacts', end_time=1637116004907, experiment_id='12', lifecycle_stage='active', run_id='3f1a65e9d6da477c892609764492217f', run_uuid='3f1a65e9d6da477c892609764492217f', start_time=1637115977957, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/322c704040874396b44a478d6ecbfc2/artifacts', end_time=1637114647594, experiment_id='12', lifecycle_stage='active', run_id='322c704040874396b44a478d6ecbfc2', run_uuid='322c704040874396b44a478d6ecbfc2', start_time=1637114622196, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/b22884d8b3264f62863a26864b68cd2c/artifacts', end_time=1637113042363, experiment_id='12', lifecycle_stage='active', run_id='b22884d8b3264f62863a26864b68cd2c', run_uuid='b22884d8b3264f62863a26864b68cd2c', start_time=1637112974713, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/83eac60d48404ea592402a1b9e7d31f9/artifacts', end_time=1637112961379, experiment_id='12', lifecycle_stage='active', run_id='83eac60d48404ea592402a1b9e7d31f9', run_uuid='83eac60d48404ea592402a1b9e7d31f9', start_time=1637111661864, status='FINISHED', user_id='karpov'>,
<RunInfo: artifact_uri='s3://kc-mlflow/12/4c862789e5c74621b1b27a9f0c31cdd0/artifacts', end_time=1637111651226, experiment_id='12', lifecycle_stage='active', run_id='4c862789e5c74621b1b27a9f0c31cdd0', run_uuid='4c862789e5c74621b1b27a9f0c31cdd0', start_time=1637111375608, status='FINISHED', user_id='karpov'>]
```

Каждый элемент полученного массива – это объект `run_info`, который описывает каждый запуск эксперимента.

Аналогичен таблице `run-ov` в ML Flow.

The screenshot shows the MLflow Experiments page for the 'PySpark-ML' experiment. The table displays 11 runs with columns for Start Time, Duration, User, Source, Models, Metrics (accuracy, f1), Parameters (Impurity, MaxDepth), and stage_3_label. The runs are sorted by Start Time, showing a progression from 5 days ago to 11 days ago. The 'Survived' status is consistent across all runs.

Start Time	Duration	User	Source	Models	accuracy	f1	Impurity	MaxDepth	stage_3_label
5 days ago	26.4s	karpov	ipykernel_...	spark-mode.../26	0.834	0.832	0.0	5	Survived
5 days ago	8.9min	karpov	ipykernel_...	spark-mode.../25	0.826	0.824	0.0	5	Survived
6 days ago	28.1s	karpov	ipykernel_...	spark-mode.../24	0.824	0.821	0.0	5	Survived
7 days ago	25.1s	karpov	ipykernel_...	spark-mode.../23	0.823	0.815	0.0	5	Survived
8 days ago	26.6s	karpov	ipykernel_...	spark-mode.../22	0.822	0.818	0.0	5	Survived
11 days ago	25.3s	karpov	ipykernel_...	spark-mode.../19	0.826	0.82	0.0	5	Survived
11 days ago	26.9s	karpov	ipykernel_...	spark-mode.../18	0.869	0.867	0.0	5	Survived
11 days ago	25.4s	karpov	ipykernel_...	spark-mode.../17	0.894	0.892	0.0	5	Survived
11 days ago	1.1min	karpov	ipykernel_...	spark-mode.../16	0.814	0.808	0.0	5	Survived
11 days ago	21.7min	karpov	ipykernel_...	-	0.843	0.837	0.0	5	Survived
11 days ago	4.6min	karpov	ipykernel_...	-	0.851	0.848	0.0	5	Survived

Посмотрим, из чего состоит один из таких объектов.

```
run_info = client.list_run_infos(exp.experiment_id)[0]
run_inforun_info = client.list_run_infos(exp.experiment_id)[0]
run_info
```

```
run_info = client.list_run_infos(exp.experiment_id)[0]
run_info
```

```
<RunInfo: artifact_uri='s3://kc-mlflow/12/1ce9927f22a14719a2783875e4a159a6/artifacts', end_time=1637596278801, experiment_id='12', lifecycle_stage='active', run_id='1ce9927f22a14719a2783875e4a159a6', run_uuid='1ce9927f22a14719a2783875e4a159a6', start_time=1637596252449, status='FINISHED', user_id='karpov'>
```

Для того, чтобы получить более подробную информацию, нам нужно забрать из `run_info` идентификатор `run`-а и вызвать функцию `get_run`.

```
run_id = run_info.run_id
run = client.get_run(run_id)
run
```

```
run_id = run_info.run_id
```

```
run = client.get_run(run_id)
run
```

```
<Run: data=<RunData: metrics={'accuracy': 0.8342541436464088, 'f1': 0.8319889038097847}, params={'Impurity': '0.0',
'MaxDepth': '5',
'MaxNumTrees': '32',
'stage_0_input': 'Sex',
'stage_0_output': 'Sex_index',
'stage_0_type': 'StringIndexerModel: uid=StringIndexer_94c0acf948b2, '
'handleInvalid=error',
'stage_1_input': 'Embarked',
'stage_1_output': 'Embarked_index',
'stage_1_type': 'StringIndexerModel: uid=StringIndexer_24bbdb026aab, '
'handleInvalid=error',
'stage_2_input': '['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Family_Size', "
"Embarked_index", 'Sex_index"]',
'stage_2_output': 'features',
'stage_2_type': 'VectorAssembler_a58911a1b434',
'stage_3_features': 'features',
'stage_3_label': 'Survived',
'stage_3_type': 'RandomForestClassificationModel: '
'uid=RandomForestClassifier_133e673a5a2d, numTrees=20, '
'numClasses=2, numFeatures=8'}, tags={'mlflow.log-model.history': '[{"run_id": "1ce9927f22a14719a2783875e4a159a6", '
'artifact_path': "spark-model", '
'utc_time_created": "2021-11-22 '
'15:51:07.090670", "flavors": {"spark": '
'{"pyspark_version": "3.1.2", "model_data": '
'"sparkml"}, "python_function": {"loader_module": '
'"mlflow.spark", "python_version": "3.8.10", '
'"data": "sparkml", "env": "conda.yaml"}}}]',
'mlflow.source.name': '/nfs/env/lib/python3.8/site-packages/ipykernel_launcher.py',
'mlflow.source.type': 'LOCAL',
'mlflow.user': 'karpov'}>, info=<RunInfo: artifact_uri='s3://kc-mlflow/12/1ce9927f22a14719a2783875e4a159a6/artifacts', end_time=1637596278801, experiment_id='12', lifecycle_stage='active', run_id='1ce9927f22a14719a2783875e4a159a6', run_uuid='1ce9927f22a14719a2783875e4a159a6', start_time=1637596252449, status='FINISHED', user_id='karpov'>>
```

Получаем подробную информацию о запуске эксперимента (в том числе залогированные параметры).

Можем обратиться к какому-нибудь определённом параметру (например, узнать значения метрик).

```
run.data.metrics
```

```
run.data.metrics
```

```
{'accuracy': 0.8342541436464088, 'f1': 0.8319889038097847}
```

Или узнать параметры `stage`-ей.

```
run.data.params
```

```
run.data.params
```

```
{'stage_1_type': 'StringIndexerModel: uid=StringIndexer_24bbdb026aab, handleInvalid=error',
 'stage_2_input': "['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Family_Size', 'Embarked_index', 'Sex_index']",
 'stage_3_type': 'RandomForestClassificationModel: uid=RandomForestClassifier_133e673a5a2d, numTrees=20, numClasses=2, numFeatures=8',
 'stage_3_label': 'Survived',
 'MaxNumTrees': '32',
 'Impurity': '0.0',
 'stage_0_type': 'StringIndexerModel: uid=StringIndexer_94c0ac948b2, handleInvalid=error',
 'stage_0_output': 'Sex_index',
 'stage_1_output': 'Embarked_index',
 'stage_0_input': 'Sex',
 'stage_1_input': 'Embarked',
 'stage_2_type': 'VectorAssembler_a58911a1b434',
 'stage_2_output': 'features',
 'stage_3_features': 'features',
 'MaxDepth': '5'}
```

> Models

Поработаем теперь с разделом Models. Зарегистрировано 3 модели.

mlflow Experiments Models GitHub Docs

Registered Models

Share and manage machine learning models. [Learn more](#)

Create Model

Search Filter Clear

Name	Latest Version	Staging	Production	Last Modified	Tags
a.savchenko	Version 8	—	Version 5	2021-11-23 20:10:36	—
car-accident-predictor	Version 4	—	—	2021-11-22 19:11:49	—
spark-model	Version 26	—	Version 25	2021-11-22 18:51:18	—

< Page 1 > 10 / page

Загрузим список зарегистрированных моделей.

```
last_models = client.list_registered_models()
```

```
client.list_registered_models()
```

```
[<RegisteredModel: creation_timestamp=1637601159393, description='', last_updated_timestamp=1637687436882, latest_versions=[<ModelVersion: creatio
n_timestamp=1637601159976, current_stage='Archived', description='', last_updated_timestamp=1637674821924, name='a.savchenko', run_id='fe578a44f72
d4a5a91e79f9078dec2e9', run_link='', source='s3://kc-mlflow/20/fe578a44f72d4a5a91e79f9078dec2e9/artifacts/a.savchenko', status='READY', status_mes
sage='', tags={}, user_id='', version='1'>,
<ModelVersion: creation_timestamp=1637687436882, current_stage='None', description='', last_updated_timestamp=1637687436882, name='a.savchenko',
run_id='624de14e9b0645e9ab7f7c2ab08f8acd', run_link='', source='s3://kc-mlflow/20/624de14e9b0645e9ab7f7c2ab08f8acd/artifacts/a.savchenko', status
='READY', status_message='', tags={}, user_id='', version='8'>,
<ModelVersion: creation_timestamp=1637615859050, current_stage='Production', description='', last_updated_timestamp=1637674273351, name='a.savch
enko', run_id='5b1a4525fb1448abaf909f784f201678', run_link='', source='s3://kc-mlflow/20/5b1a4525fb1448abaf909f784f201678/artifacts/a.savchenko',
status='READY', status_message='', tags={}, user_id='', version='5'>], name='a.savchenko', tags={}>,
<RegisteredModel: creation_timestamp=1637595938544, description='', last_updated_timestamp=1637597509894, latest_versions=[<ModelVersion: creatio
n_timestamp=1637597509894, current_stage='None', description='', last_updated_timestamp=1637597509894, name='car-accident-predictor', run_id='22b4
5a1486a2486ab2f8ff2eb89a1c6', run_link='', source='s3://kc-mlflow/19/22b45a1486a2486ab2f8ff2eb89a1c6/artifacts/car-accident-predictor', status
='READY', status_message='', tags={}, user_id='', version='4'>], name='car-accident-predictor', tags={}>,
<RegisteredModel: creation_timestamp=1637106829432, description='', last_updated_timestamp=1637596278768, latest_versions=[<ModelVersion: creatio
n_timestamp=1637596278768, current_stage='None', description='', last_updated_timestamp=1637596278768, name='spark-model', run_id='1ce9927f22a1471
9a2783875e4a159a6', run_link='', source='s3://kc-mlflow/12/1ce9927f22a14719a2783875e4a159a6/artifacts/spark-model', status='READY', status_message
='', tags={}, user_id='', version='26'>,
<ModelVersion: creation_timestamp=1637359515018, current_stage='Archived', description='', last_updated_timestamp=1637493760891, name='spark-mod
el', run_id='109b758056cb48b5b9ad92d0d70e3566', run_link='', source='s3://kc-mlflow/12/109b758056cb48b5b9ad92d0d70e3566/artifacts/spark-model', st
atus='READY', status_message='', tags={}, user_id='', version='22'>,
<ModelVersion: creation_timestamp=1637493454514, current_stage='Production', description='', last_updated_timestamp=1637493760891, name='spark-m
odel', run_id='79d1b47e4ea147ac93c2ecd06060888', run_link='', source='s3://kc-mlflow/12/79d1b47e4ea147ac93c2ecd06060888/artifacts/spark-model',
status='READY', status_message='', tags={}, user_id='', version='25'>], name='spark-model', tags={}>]
```

Получили список объектов RegisteredModel.

Выберем одну модель и посмотрим какие у неё есть последние версии.

```
reg_model = last_models[0]
reg_model
reg_model.latest_versions
```



```
reg_model.latest_versions
```

```
[<ModelVersion: creation_timestamp=1637601159976, current_stage='Archived', description='', last_updated_timestamp=1637674821924, name='a.savchenko', run_id='fe578a44f72d4a5a91e79f9078dec2e9', run_link='', source='s3://kc-mlflow/20/fe578a44f72d4a5a91e79f9078dec2e9/artifacts/a.savchenko', status='READY', status_message='', tags={}, user_id='', version='1'>, <ModelVersion: creation_timestamp=1637687436882, current_stage='None', description='', last_updated_timestamp=1637687436882, name='a.savchenko', run_id='624de14e9b0645e9ab7f7c2ab08f8acd', run_link='', source='s3://kc-mlflow/20/624de14e9b0645e9ab7f7c2ab08f8acd/artifacts/a.savchenko', status='READY', status_message='', tags={}, user_id='', version='8'>, <ModelVersion: creation_timestamp=1637615859050, current_stage='Production', description='', last_updated_timestamp=1637674273351, name='a.savchenko', run_id='5b1a4525fb1448abaf909f784f201678', run_link='', source='s3://kc-mlflow/20/5b1a4525fb1448abaf909f784f201678/artifacts/a.savchenko', status='READY', status_message='', tags={}, user_id='', version='5'>]
```

Получили список из 3х моделей: последняя версия из архива, последняя версия без назначенного stage-а и последняя версия, которая помечена тегом Production. Получаем последнюю версию по каждому stage-у.

Напишем функцию, которая по имени зарегистрированной модели будет находить последнюю модель в stage Production.

```
def get_last_prod_model(name):
    last_models = client.get_registered_model(name).latest_versions
    models = list(filter(lambda x: x.current_stage == 'Production', last_models))
    if len(models) == 0:
        return None
    else:
        return models[0]
```

Получим такую модель.

```
model_version = get_last_prod_model('spark-model')
model_version
```

```
def get_last_prod_model(name):
    last_models = client.get_registered_model(name).latest_versions
    models = list(filter(lambda x: x.current_stage == 'Production', last_models))
    if len(models) == 0:
        return None
    else:
        return models[0]
```

```
model_version = get_last_prod_model('spark-model')
model_version
```

```
<ModelVersion: creation_timestamp=1637493454514, current_stage='Production', description='', last_updated_timestamp=1637493760891, name='spark-model', run_id='79d1b47e4ea147ac93c2ecdbc0660888', run_link='', source='s3://kc-mlflow/12/79d1b47e4ea147ac93c2ecdbc0660888/artifacts/spark-model', status='READY', status_message='', tags={}, user_id='', version='25'>
```

Узнаем версию этой модели.

```
model_version.version
```

```
model_version.version
```

```
'25'
```

> Load model

Также мы можем загружать модели. Для этого нужна spark сессия.

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



```
SparkSession - in-memory
SparkContext
Spark UI
Version      v3.1.2
Master       local[*]
AppName      PySparkMLFlowClient
```

Рассмотрим 2 способа загрузки моделей.

Первый – это через имя модели.

```
from pyspark.sql.functions import *
model = mlflow.spark.load_model(f'models:/spark-model/{model_version.version}')
```

В ходе выполнения кода ML Flow Client обращается к ML Flow, проверяет есть ли там модель с такими именем и версией, после чего найденная модель загружается в spark сессию.

Посмотрим на нашу модель.

```
type(model)
```

```
type(model)
pyspark.ml.pipeline.PipelineModel
```

Другой вариант загрузки – загрузка из эксперимента. Обращаемся к запускам, указывая run_id.

```
model = mlflow.spark.load_model(f'runs:{run_id}/spark-model')
type(model)
```

После загрузки модель можно применять к нашим данным.