|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра прикладной математики (ПМ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Системы управления данными»

**Практическое занятие № 9**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИМБО-02-22, Ким Кирилл Сергеевич* | (подпись) | |
| Преподаватель | *Алексеева Екатерина Сергеевна, преподаватель* | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_202\_\_г. | |  | |

Москва 2025 г.

1. Практическая работа №9
   1. Цель работы

### В данной лабораторной работе будет показано, как создавать модель логистической регрессии в кластере Spark. В работе будет использоваться блокнот Jupyter для создания и тестирования приложения.

## Извлеките необходимые типы данных (Vectors, LogisticRegression).

### Подготовите тренировочные данные из списка кортежей.

﻿from pyspark.ml.linalg import Vectors

from pyspark.ml.classification import LogisticRegression

#тренировочные данные со свойствами и метками

﻿training = spark.createDataFrame([

(1.0, Vectors.dense([0.0, 1.1, 0.1])),

(0.0, Vectors.dense([2.0, 1.0, -1.0])),

(0.0, Vectors.dense([2.0, 1.3, 1.0])),

(1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "features"])

## Создание модели и проверка тренировочных данных.

### Создание экземпляра и модели LogisticRegresion с параметрами, хранимыми в lr.

﻿lr = LogisticRegression(maxIter=10, regParam=0.01)

Model1=lr.fit(training)

Примечание: экземпляр LogisticRegresion – это оценщик.

### Выведите тренировочные данные.

﻿training.show(10,False)

### Должен получиться похожий вывод:

###### 

## ﻿Создание новой модели.

### Новая модель использует параметры paramMapCombined. paramMap будет использовать словарь Python в качестве параметров.

paramMap = {lr.maxIter: 20}

paramMap[lr.maxIter] = 30 # Определение 1 параметра, перезаписывающего maxIter.

# Определение множественных параметров.

paramMap.update({lr.regParam: 0.1, lr.threshold: 0.55})

﻿paramMap2 = {lr.probabilityCol: "myProbability"} # type: ignore

paramMapCombined = paramMap.copy()

paramMapCombined.update(paramMap2) # type: ignore

# paramMapCombined перезаписывает параметры,

# установленные ранее через методы lr.set\*.

model2 = lr.fit(training, paramMapCombined)

## Подготовьте данные для использования на обученной модели. Сделайте прогноз на основе данных.

﻿﻿ ﻿test = spark.createDataFrame([

(1.0, Vectors.dense([-1.0, 1.5, 1.3])),

(0.0, Vectors.dense([3.0, 2.0, -0.1])),

(1.0, Vectors.dense([0.0, 2.2, -1.5]))], ["label", "features"])

# Получите прогноз, используя метод Transformer.transform().

# LogisticRegression.transform необходимы только столбцы со свойствами.

prediction = model2.transform(test)

result = prediction.select("features", "label", "myProbability", "prediction").collect()

﻿for row in result:

print("features=%s, label=%s -> prob=%s, prediction=%s"

% (row.features, row.label, row.myProbability, row.prediction))

### Вы можете наблюдать схожий вывод:

###### 

# Создание приложения для кластеризации

В данной работе будет показано, как создать алгоритм кластеризации KMeans в кластере Spark.

## Импорт типов данных, необходимых в данной работе lab.

### Вставьте код снизу в ячейку и нажмите SHIFT + ENTER. SHIFT + ENTER запускает выполнение кода в текущей ячейке и переходит к следующей.

﻿from pyspark.ml.clustering import KMeans

from pyspark.ml.evaluation import ClusteringEvaluator

## Загрузите пример входных данных и сохраните их в памяти.

﻿#загрузка данных.

dataset = spark.read.format("libsvm").load("file:///home/student/Data/sample\_kmeans\_data.txt")

### Проверьте тренировочные данные.

﻿dataset.show(20,False)

### Вы можете наблюдать схожий вывод:

###### 

## Создайте модель KMeans со столбцами свойств и столбцом меток

﻿# Тренировка модели k-means.

kmeans = KMeans().setK(2).setSeed(1)

model = kmeans.fit(dataset)

## Получение прогноза с помощью модели.

### Для этого будет использоваться метод transform созданной модели.

﻿# Создание прогноза

predictions = model.transform(dataset)

## 

## Запуск на тестовых данных

### На основе модифицированных входных данных, подготовим алгоритм кластеризации.

### Получается, что вы обучаете и тестируете модель на одном и том же набором данных. Тестовый набор данных на самом деле должен быть другим, чтобы проверить точность модели.

### Оцените кластеризацию, вычислив коэффициент силуэта

evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)

print("Silhouette with squared euclidean distance = " + str(silhouette))

### Выведите результат.

centers = model.clusterCenters()

print("Cluster Centers: ")

for center in centers:

print(center)

#### Должен получиться вывод, похожий на этот:

#### 

# Создание модели линейной регрессии

### В этой работе будет показано, как построить модель линейной регрессии в кластере Spark. Будет использоваться блокнот Jupyter для сборки и тестирования приложения.

Набор данных Boston Housing — это популярный набор данных машинного обучения на Kaggle. Набор данных получен из информации, собранной Службой переписи населения США о жилье в Бостоне, штат Массачусетс. Набор данных содержит следующее:

* CRIM - уровень преступности на душу населения по городам
* ZN - доля земли под жилую застройку, зонированная под участки более 25 000 кв. футов.
* INDUS - доля акров неторгового бизнеса в расчете на один город.
* CHAS - фиктивная переменная реки Чарльз (1, если участок ограничивает реку; 0 в противном случае)
* NOX - концентрация оксидов азота (частей на 10 миллионов)
* RM - среднее количество комнат в доме
* AGE - доля занимаемых владельцами единиц, построенных до 1940 г.
* DIS - взвешенные расстояния до пяти бостонских центров занятости
* RAD - индекс доступности радиальных автомобильных дорог
* TAX - полная ставка налога на имущество за 10 000 долларов США
* PTRATIO - соотношение учеников и учителей по городам
* B - 1000(Bk - 0.63)^2 где Bk - доля черных по городам
* LSTAT - % более низкого статуса населения
* MEDV - Средняя стоимость домов, занимаемых владельцами, в 1000 долларов США

### Задача: предсказать MEDV, среднюю стоимость домов, занимаемых владельцами.

### 

### 

## Формирование исходного DataFrame

### Скопируйте housing.csv из /home/student/Data в директорию локальной HDFS

### Используйте spark.read() для создания DataFrame из housing.csv. Исходный .csv файл содержит строку заголовков. Убедитесь, что выставили параметр для чтения заголовка. Кроме того, установите параметр для вывода схемы (inferSchema). Проверьте схему полученного DataFrame. Соответствует ли она тому, что представлено ниже?

* CRIM - double
* ZN - double
* INDUS - double
* CHAS - integer
* NOX - double
* RM - double
* AGE - double
* DIS - double
* RAD - integer
* TAX - integer
* PTRATIO - double
* B - double
* LSTAT - double
* MEDV - double

### 

### К сожалению, Spark неправильно определил схему, поэтому её вручную создать вручную.

#### Создайте вышеприведённую схему в помощью SructType() и StructField(). Не забудьте импортировать необходимые SQL типы из pyspark.sql.types

#### Исследуйте dataframe, используя printSchema(). Убедитесь, что на этот раз схема корректная. Вывод должен соответствовать приведённому ниже.

###### 

### Выведите 20 строк, убедившись, что усечение отключено. Вывод должен выглядеть примерно так, как показано ниже:

###### 

## Предобработка данных

### Обратите внимание на приведенную выше распечатку. Несколько строк содержат нулевые значения для столбца CHAS. Этот столбец равен 1, если недвижимость граничит с рекой Чарльз, и 0 в противном случае. Кажется, нет хорошего способа заполнить эту информацию из других строк. В этом случае нам придется удалить все строки, содержащие нулевое значение. Для этого существует команда Spark:

dataframe.na.drop()

### 

### Создайте список, содержащий имена всех столбцов кроме MEDV. MEDV – это колонка меток.

#### Можно использовать любой метод, который понравится, включая ручное создание списка. Однако имейте в виду, что API dataframe.columns() возвращает список всех столбцов. Если вы распечатаете список, вывод должен быть похож на приведенный ниже:

###### 

### Создайте трансформер чтобы собрать столбцы объектов в виде Vector Column. Чтобы выполнить это воспользуйтесь pyspark.ml.feature.VectorAssembler() API.

### <https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.ml.feature.VectorAssembler.html>

from pyspark.ml.feature import VectorAssembler

assembler = VectorAssemble(inputCols=<list of feature column names>,

outputCol=<name of new vector column>)

### Теперь воспользуйтесь методом transform(<dataframe>), чтобы превратить исходный DataFrame в новый. В полученном DataFrame появится дополнительный столбец с именем, определённым выше в параметре outputCol. Выведите преобразованный DataFrame. Это должно выглядеть примерно так, как показано ниже:

###### 

### Разделение данных на тренировочный и тестовый наборы.

#### В Spark MLlib существует отличный метод, случайно разделяющий наборы данных. Вызывается с помощью: dataframe.randomSplit(<list of doubles as weigh>). Используйте веса 0.7 и 0.3, чтобы 70% данных попали в тренировочный набор, а оставшиеся 30% в тестовый.

weights = [0.7, 0.3]

train, test = df.randomSplit(weights)

#### Выведите по несколько строчек тренировочного и тестового набора. Каждый из наборов – это DataFrame. Заодно проверьте количество строк в каждом из наборов.

## 

## Создание модели линейной регрессии

### Используйте класс LinearRegression из pyspark.ml.regression для создания нового оценщика линейной регрессии. Обратитесь к синтаксису ниже. <https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.ml.regression.LinearRegression.html?highlight=linearregression#pyspark.ml.regression.LinearRegression> В данной работе outputCol, который был создан на шаге 2.3, будет называться «featuresCol». Это столбец признаков, который был добавлен работой VectorAssembler. labelCol – это MEDV. Это та самая средняя стоимость домов, занимаемых владельцами, которую надо предсказать.

### 

### Воспользуйтесь методом оценщика fit(<training set dataframe>). Это создаст новую модель линейной регрессии на основе тренировочного датасета.

## Тестирование модели линейной регрессии.

### Воспользуйтесь методом transform(<test dataframe>) у созданной выше модели для получения прогноза. Для этого понадобится DataFrame созданный в п. 2.5.

### 

### Выведите несколько строк созданного DataFrame с прогнозом. Вывод должен быть похож на тот, что ниже:

###### 테이블이(가) 표시된 사진 자동 생성된 설명