Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Отчет по лабораторной работе №3 Microsoft ML Studio и Microsoft Cognitive Services по дисциплине

«Инструменты бизнес-аналитики»

ИСПОЛНИТЕЛЬ:

Березин И.С.

Группа ИУ5-43М

_____

"__"_____2020 г.

Москва 2020

1. Цель лабораторной работы

Целью данной ЛР является получение необходимых знаний и навыков для работы в среде Microsoft Machine Learning Studio, которая предоставляет разработчикам и специалистам по обработке и анализу данных широкий спектр возможностей для быстрой разработки, обучения и развертывания моделей машинного обучения.

2. Практическая часть

2.1. Машинное обучение Azure

Данный раздел лабораторной работы включает в себя проведение серии экспериментов, направленных на решение в среде Microsoft Azure ML Studio типовых задач машинного обучения: регрессии, классификации и кластеризации. В этом разделе мы будем использовать образец набора данных «Данные электростанции комбинированного типа». Набор данных содержит 9568 точек данных, собранных с электростанции с комбинированным циклом за 6 лет (2006-2011 гг.), в том числе информацию о среднечасовых температурах, давления окружающей среды, относительной влажности и чистой почасовой выработки электроэнергии.

2.1.1. Эксперимент №1 – Регрессия

Шаг 1: Импортировать набор данных для регрессии в ML Studio. DATASETS => +NEW =>

Шаг 2: Создать новый эксперимент. EXPERIMENTS => +NEW => Blank Experiment

Шаг 3: Переименовать эксперимент. "Experiment created on.." => "Regression"

Шаг 4: Добавить импортированный набор данных. Перетащить ранее загруженный датасет Folds5x2_pp.csv в область эксперимента.

Шаг 5: Визуализировать данные. Выбрать Visualize.

Шаг 6: Добавить описание к эксперименту из датасета.

Шаг 7:

1) Переименовать столбцы датасета. Перетащить объект Edit Metadata в область экспиримента. Search.. => Edit Metadata => Lauch columns selector;
2) Привести значения в датасете к типу Integer Data Type => "Integer";
3) Запустите эксперимент, нажав RUN внизу страницы;
4) Визуализировать данные. Выбрать Visualize.

Шаг 8:

1) Отфильтровать датасет, выбрав наиболее влияющие признаки на прогноз чистой почасовой выработки электроэнергии (Output). Перетащить объект Filter Based Feature Selection в область эксперимента и выбрать столбец Output. Search..=> Filter Based Feature Selection => Lauch columns selector;

2) Указать необходимое количество призаков. Number of desired features => "3";

3) Выбрать статистический метрику Пирсона для начисления баллов для каждого столбца(признака). Feature scoring method => "Pearson Correlation";

4) Запустить эксперимент, нажав RUN внизу страницы;

5) Визуализировать данные. Выбрать Visualize.

Шаг 9: Разделить датасет на обучающую и тестовую выборку в соотношении 70/30. Перетащить объект Split Data в область эксперимента: Search.. => Split Data => Fraction of rows.... => "0.7"

Шаг 10:

1) Перетащить объект Linear Regression в область экспиримента. Search.. => Linear Regression;

2) Выбрать метод наименьших квадратов и его параметры. Solution method => "Ordinary Least Squares", L2 => "0.001".

Шаг 11:

1) Обучить модель. Выбрать целевой (выходной) признак. Перетащить объект Train Model в область экспиримента и столбец Output. Search.. => Train Model => Lauch columns selector => "Output";

2) Посмотреть коэффициенты каждого из признаков.

Шаг 12: Получить предсказания модели на тестовом датасете.

1) Соединить модель с обучающим датасетом и выбранным алгоритмом. Соединить Train Model с модулем Split Data и модулем Linear Regression;

2) Запустить эксперимент, нажав RUN внизу страницы;

3) Визуализировать данные. Выбрать Visualize.

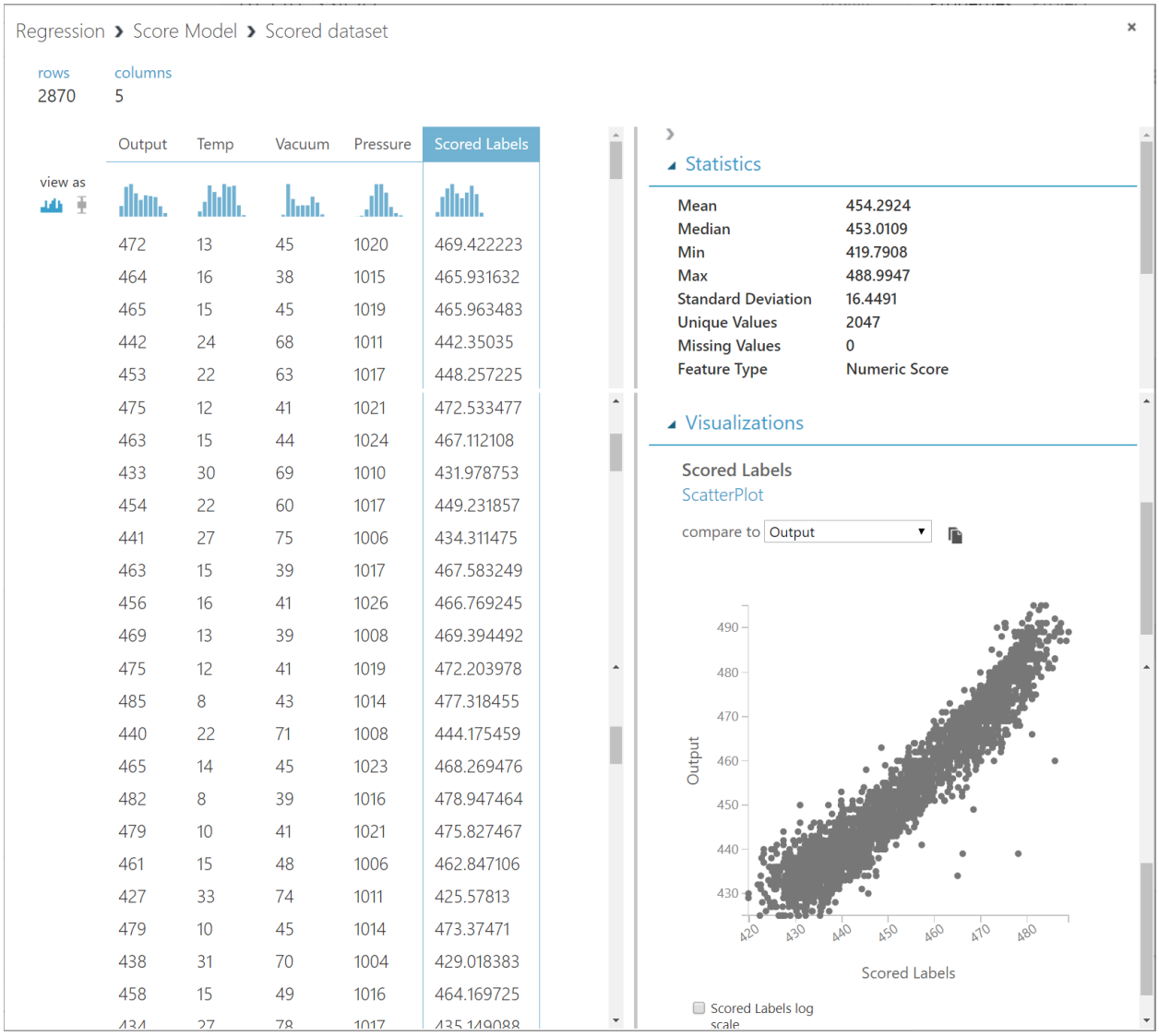
Шаг 13: Оценить модель.

1) Перетащить объект Score Model в область экспиримента. Search.. => Score Model => Lauch columns selector;

2) Запустить эксперимент, нажав RUN внизу страницы;

3) Визуализировать данные. Выбрать Visualize.

Визуализация результата:



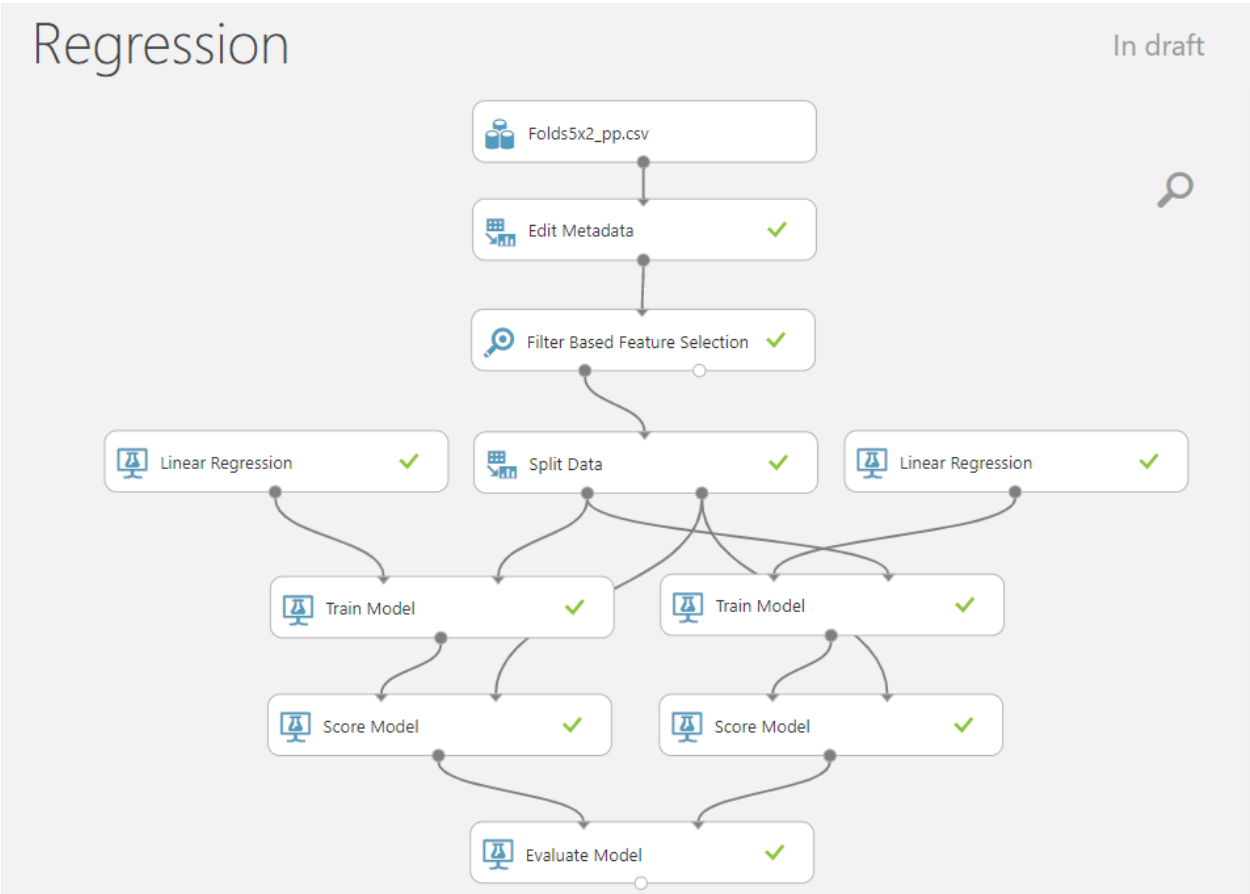
Шаг 14: Повторить шаги с 10 по 12, но в этот раз выбрать метод градиентный спуск и его параметры: размер шага, количество итераций, вес регуляризации. Solution method=> "Online Gradien Descent"; Learning rate => "0.1"; Number of training => "1000"; L2 => "0.001". Этот метод минимизирует ошибку на каждом шаге процесса обучения модели.

Шаг 15:

1) Сравнить две получившиеся модели. Перетащить объект Evaluate Model в область эксперимента. Search..=> Evaluate Model;

2) Визуализировать данные. Выбрать Visualize.

Схема эксперимента и визуализация результатов оценки моделей:



Regression › Evaluate Model › Evaluation results

Metrics

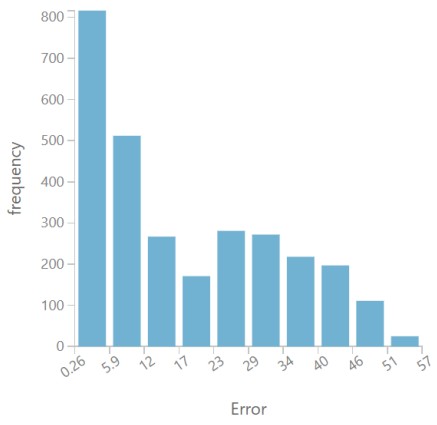
Mean Absolute Error	3.930113
Root Mean Squared Error	4.980534
Relative Absolute Error	0.265396
Relative Squared Error	0.086263
Coefficient of Determination	0.913737

Metrics

Mean Absolute Error	18.444372
Root Mean Squared Error	23.704959
Relative Absolute Error	1.245527
Relative Squared Error	1.95411
Coefficient of Determination	-0.95411

Error Histogram

Error Histogram



2.1.2. Эксперимент №2.1 – Кластеризация

Шаг 1: Создать новый эксперимент. EXPERIMENTS => +NEW => Blank Experiment

Шаг 2: Импортировать набор данных для кластеризации в ML Studio. Перетащить объект Import Data в область эксперимента. Search..=> Import Data => Data source URL: http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data => Data source: "Web URL Via HTTP", Data format: "CSV".

Шаг 3: Удалить строки в данных с пропущенными значениями с помощью модуля Clean Missing Data . Search..=> Clean Missing Data => Cleaning mode: "Remove entire row"

Шаг 4:

1) Переименовать столбцы датасета. Перетащить объект Edit Metadata в область эксперимента. Search..=> Edit Metadata => Lauch columns selector: All columns, All labels => F1,F2,F3,F4,Label;
2) Запустить эксперимент, нажав RUN внизу страницы;
3) Визуализировать данные. Выбрать Visualize.

Шаг 5:

1) Столбец Label сделать меткой. Перетащить объект Edit Metadata в область эксперимента. Search..=> Edit Metadata => Launch columns selector: Column names: "Label", Fields: "Label"
2) Запустите эксперимент, нажав RUN внизу страницы.
3) Визуализировать данные. Выбрать Visualize.

Шаг 6

Разделить датасет на обучающую и тестовую выборку в соотношении 60/40. Перетащить объект Split Data в область эксперимента.

Search..=> Split Data => Fraction of rows.... => "0.7"

Шаг 7: Перетащить объект K-Means Clustering в область экспиримента. Search..=> K-Means Clustering. Настроить параметры следующим образом:

1) Create trainer mode => "Single Parameter"
2) Number of Centroids: "2"
3) Metric: "Cosine"
4) Iterations: "10000"
5) Assign Label Mode: "Owerwrite..."

Шаг 8: Обучить модель. Выбрать столбцы для кластеризации. Перетащить объект Train Clustering Model в область экспиримента и столбец Output. Search.. => Train Clustering Model => Lauch columns selector => F1,F2,F3,F4,Label

Шаг 9: Применить алгоритм к данным и инициализируем кластеры.

1) Соединить модель с обучающими датасетом и выбранным алгоритмом. Соединить Train Clustering Model с модулем Split Data и модулем K-Means Clustering;

2) Запустить эксперимент, нажав RUN внизу страницы;

3) Визуализировать данные. Выбрать Visualize.

Визуализация результата обучения первой модели:

Шаг 10: Повторить шаги с 7 по 9. При этом поменять гиперпараметры алгоритма K-Means Clustering на следующие:

1) Create trainer mode => "Single Parameter"

2) Number of Centroids: "3"

3) Metric: "Cosine"

4) Iterations: "100"

5) Assign Label Mode: "Owerwrite..."

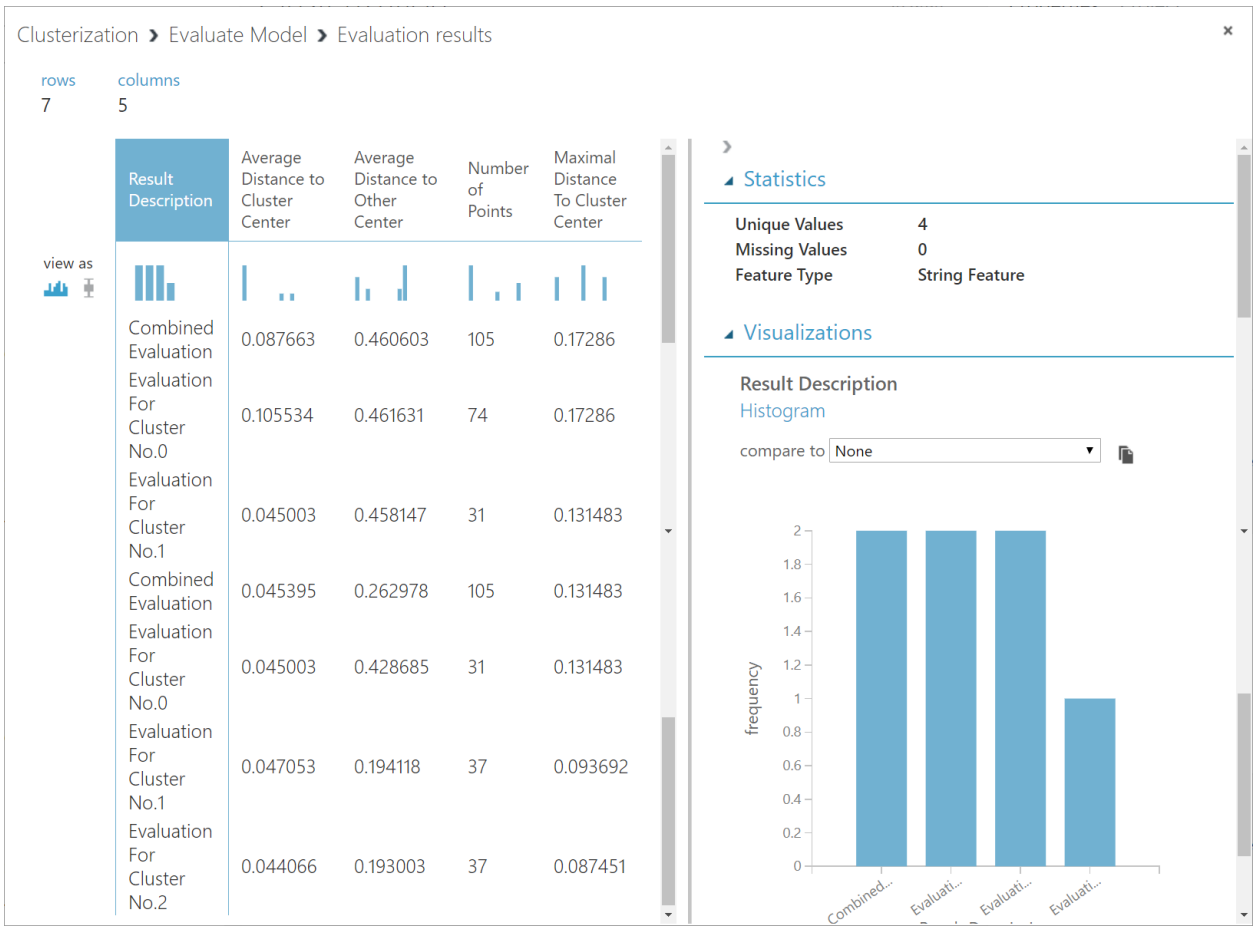
Визуализация результата обучения второй модели:

Шаг 11:

1) Сравнить две получившиеся модели. Перетащить объект Evaluate Model в область эксперимента. Search.. => Evaluate Model;

2) Визуализировать данные. Выбрать Visualize.

Визуализация результата сравнения моделей:



2.1.3. Эксперимент №2.2 – Кластеризация

Повторить шаги из прошлого эксперимента (выгрузить датасет из прошлого эксперимента с помощью модуля Convert to Dataset). На шаге 7 применить способ подбора количества центроидов с помощью следующих значений параметров алгоритма обучения модели:

1) Create trainer mode => "Parameter Range"
2) User Range Builder: "2,3,4,5"
3) Metric: "Cosine"
4) Iterations: "1000"
5) Assign Label Mode: "Owerwrite... "

На шаге 8 добавить модуль Train Clustering Model (All columns Exclude column names: Class) и параллельно (для сравнения) добавить модуль Sweep Clustering (Exclude column names: Class; Metric for measuring: "Simplified ... "; Specify parameter sweeping: "Entire grid").

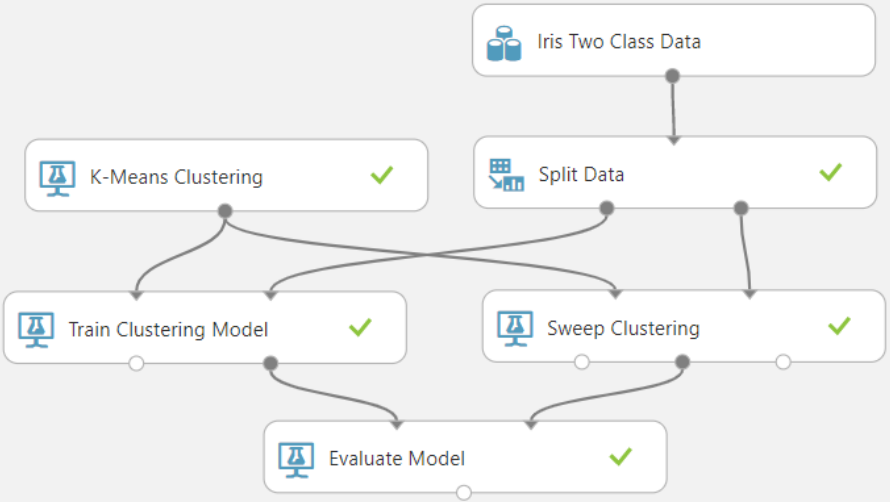
Сравнить два способа с помощью модуля Evaluate Model.

Модуль Sweep Clustering создает несколько моделей кластеризации, перечисленных в порядке точности. Модели измеряются с использованием всех возможных метрик.

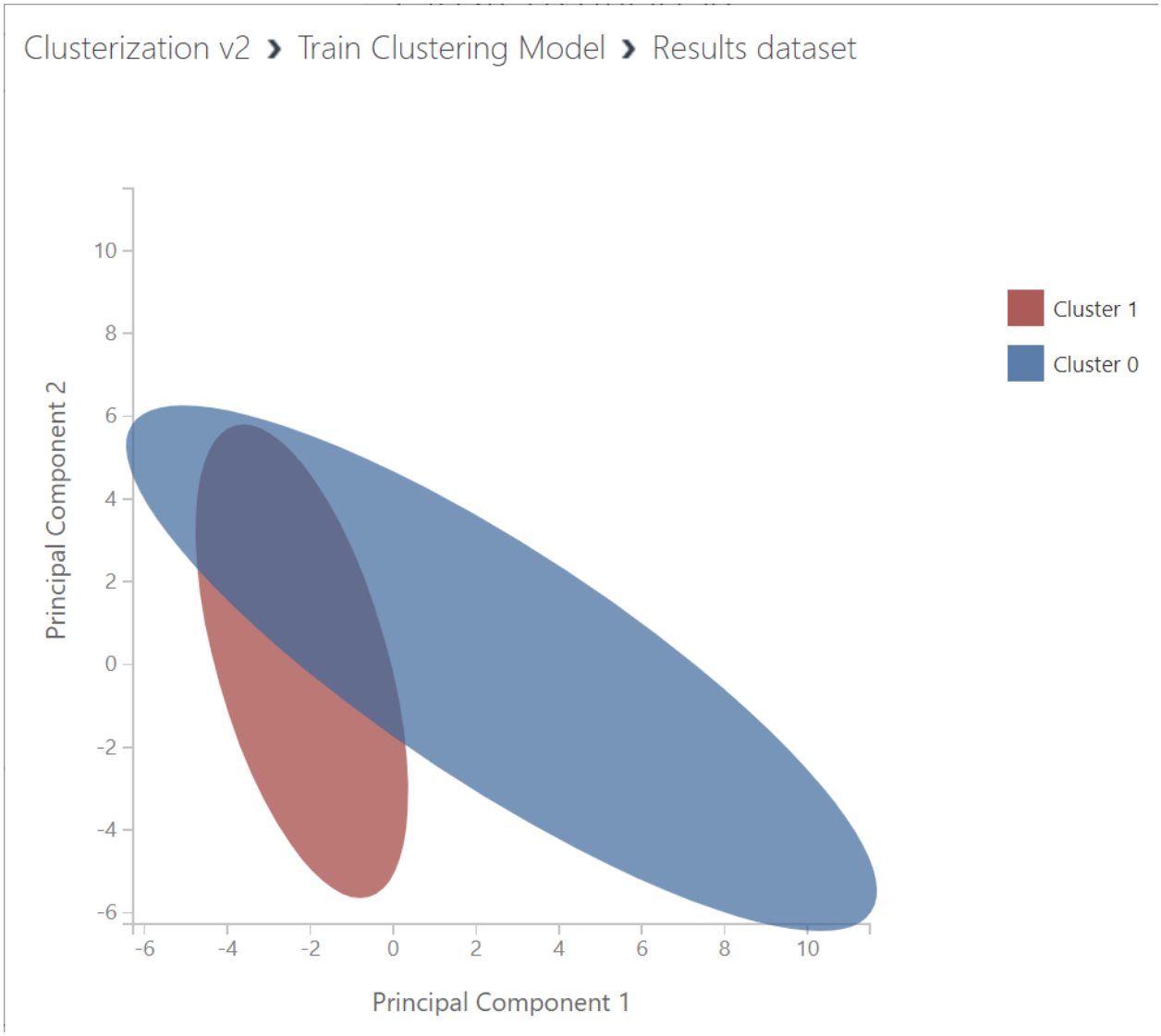
Схема эксперимента:

Clusterization v2

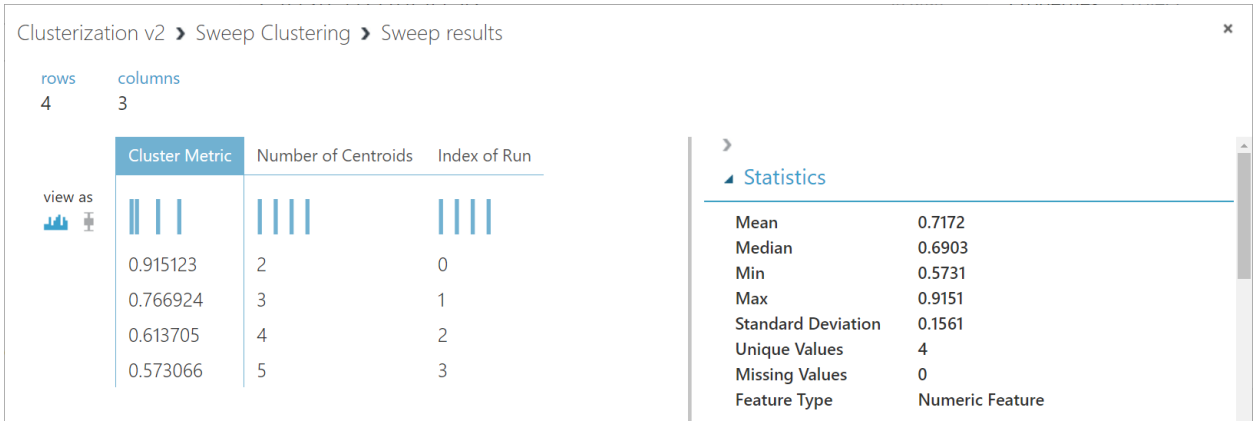
In draft



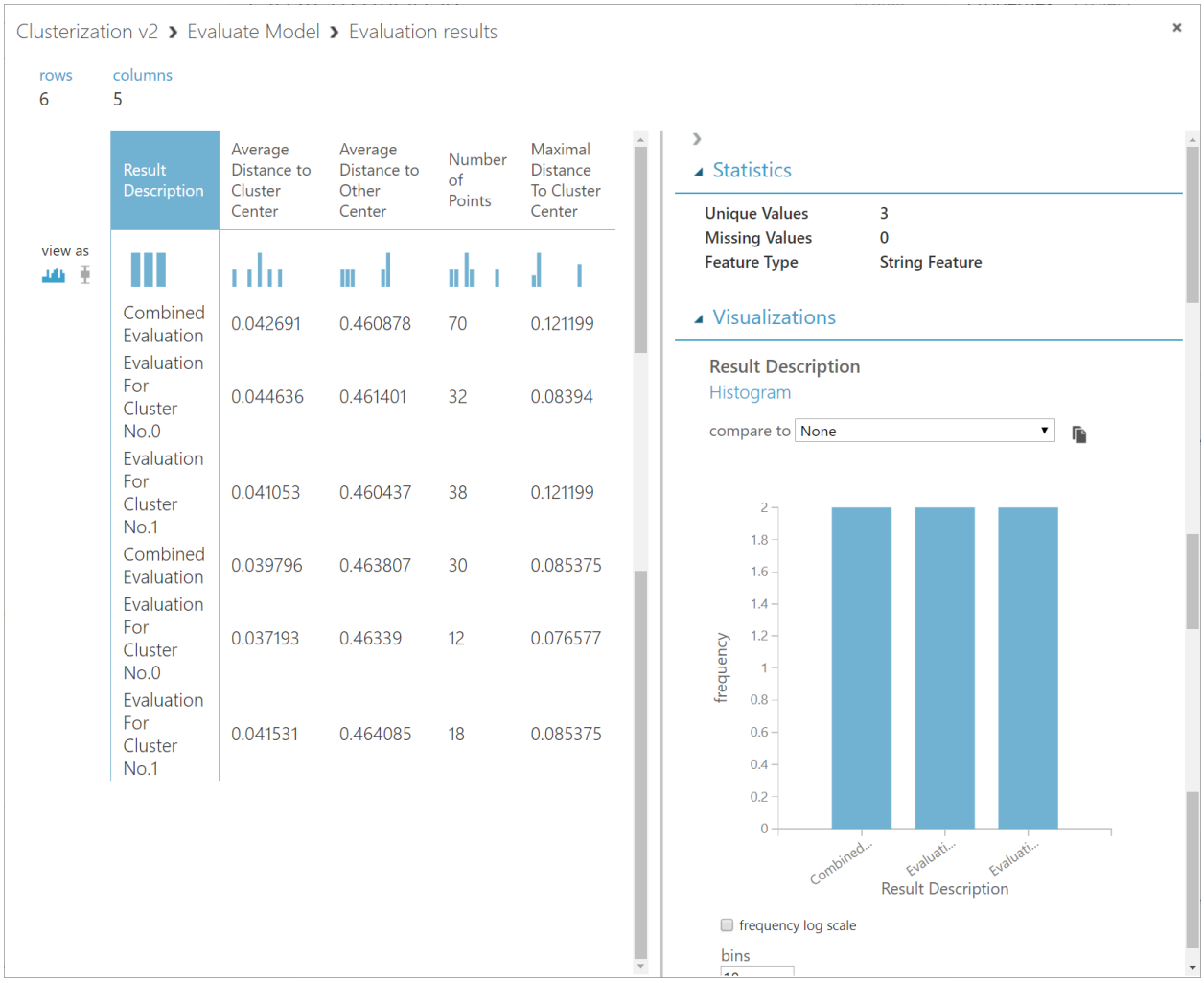
Визуализация результата обучения первой модели:



Визуализация результата работы модуля Sweep Clustering:



Визуализация результата сравнения моделей кластеризации:



2.1.4. Эксперимент №3 – Классификация

Шаг 1: Импортировать набор данных для классификации в ML Studio. DATASETS => +NEW

Шаг 2: Создать новый эксперимент. EXPERIMENTS => +NEW => Blank Experiment

Шаг 3:

1) Выбрать необходимые столбцы датасета. Перетащить объект Select Columns in Dataset в область эксперимента. Search.. => Select Columns in Dataset => Lauch columns selector => Columns names: "STG,SCG,STR,LPR,PEG,UNS"

2) Запустить эксперимент, нажав RUN внизу страницы;

3) Визуализировать данные. Выбрать Visualize.

Шаг 4:

1) Столбец UNS сделать меткой и переименовать. Перетащить объект Edit Metadata в область эксперимента. Search.. => Edit Metadata => Lauch columns selector: Column names: "UNS", Fields: "Label", New column names: "Level";

2) Запустить эксперимент, нажав RUN внизу страницы.

3) Визуализировать данные. Выбрать Visualize.

Шаг 5: Перетащить объект Multiclass Neural Network в область эксперимента. Search.. => Multiclass Neural Network и задать следующие значения:

1) Create trainer mode=> "Single Parameter"

2) Number of hidden nodes: "1000"

3) The learning rate: "0.1"

4) Number of learning iterations: "1000"

5) The initial learning: "0.1"

6) The momentum: "0"

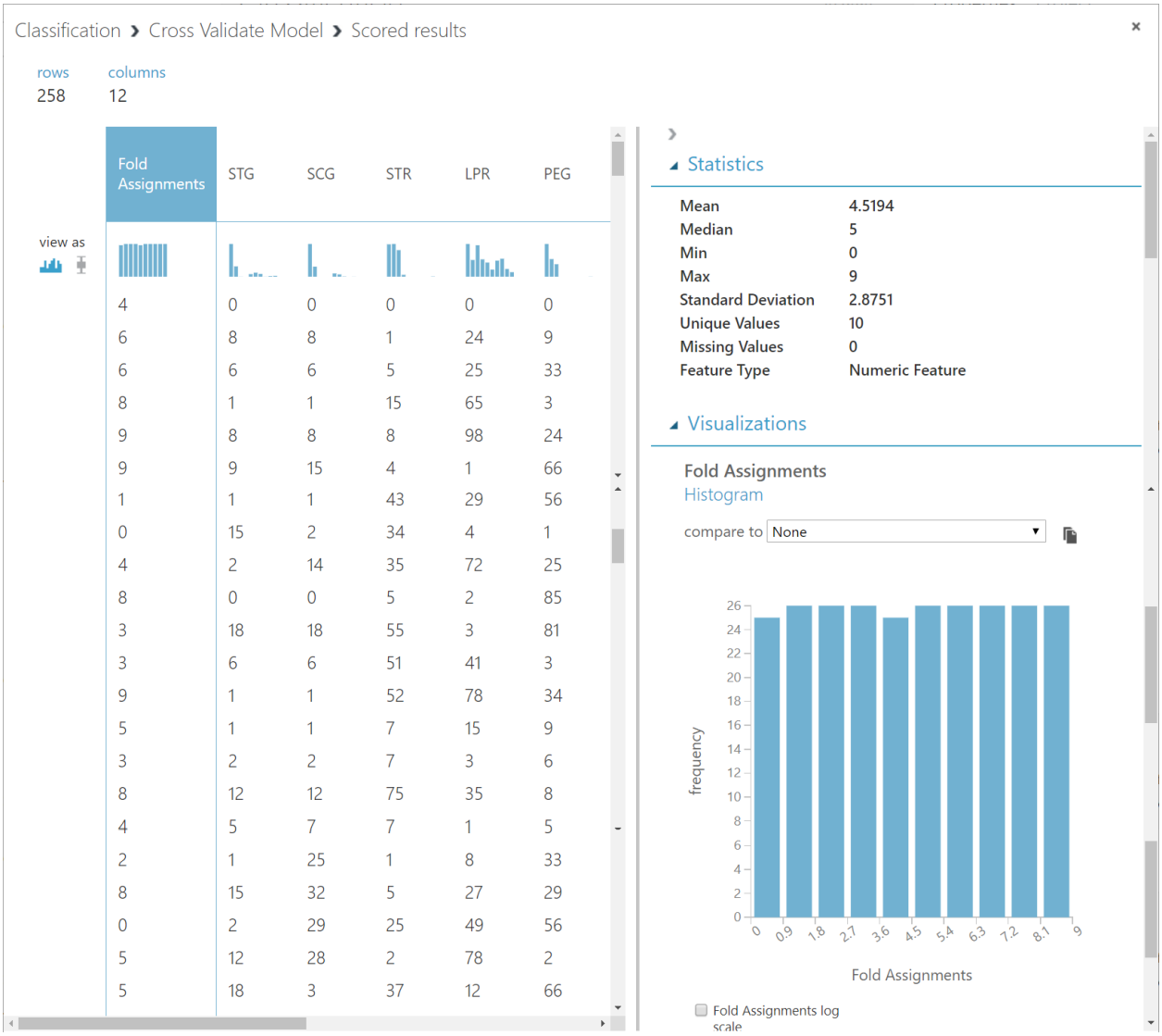
Шаг 6:

1) Перетащить объект Cross Validate Model в область эксперимента. Search.. => Cross Validate Model => Selected columns => Column names: "Level";

2) Запустить эксперимент, нажав RUN внизу страницы;

3) Визуализировать данные. Выбрать Visualize.

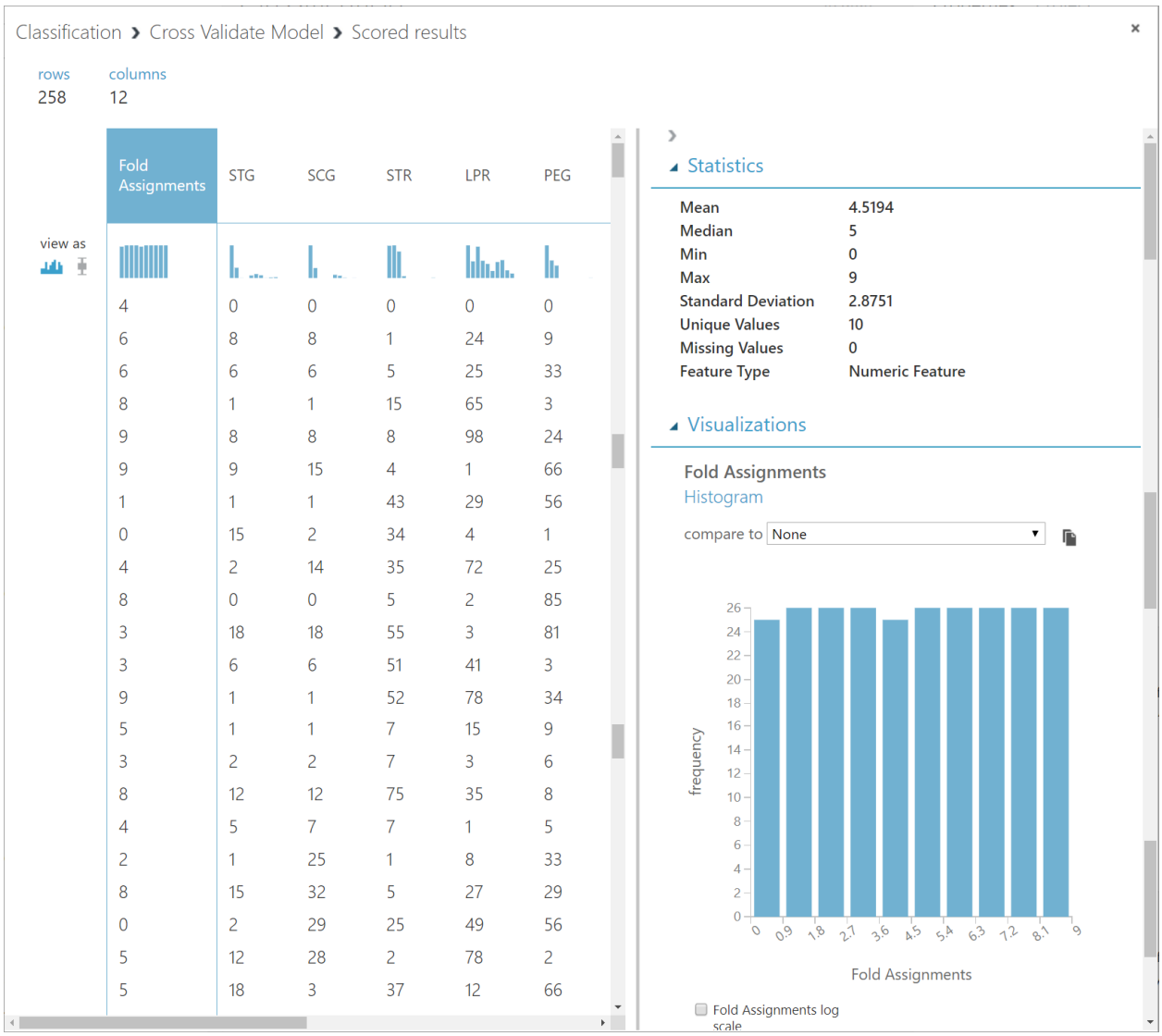
Визуализация результатов:



Шаг 7: Повторить шаги с 5 по 6, поменяв при этом гиперпараметры алгоритма Multiclass Neural Network:

1) Create trainer mode => "Single Parameter"
2) Number of hidden nodes: "100"
3) The learning rate: "0.1"
4) Number of learning iterati: "100"
5) The initial learning: "0.1"
6) The momentum: "0"

Визуализация результатов:

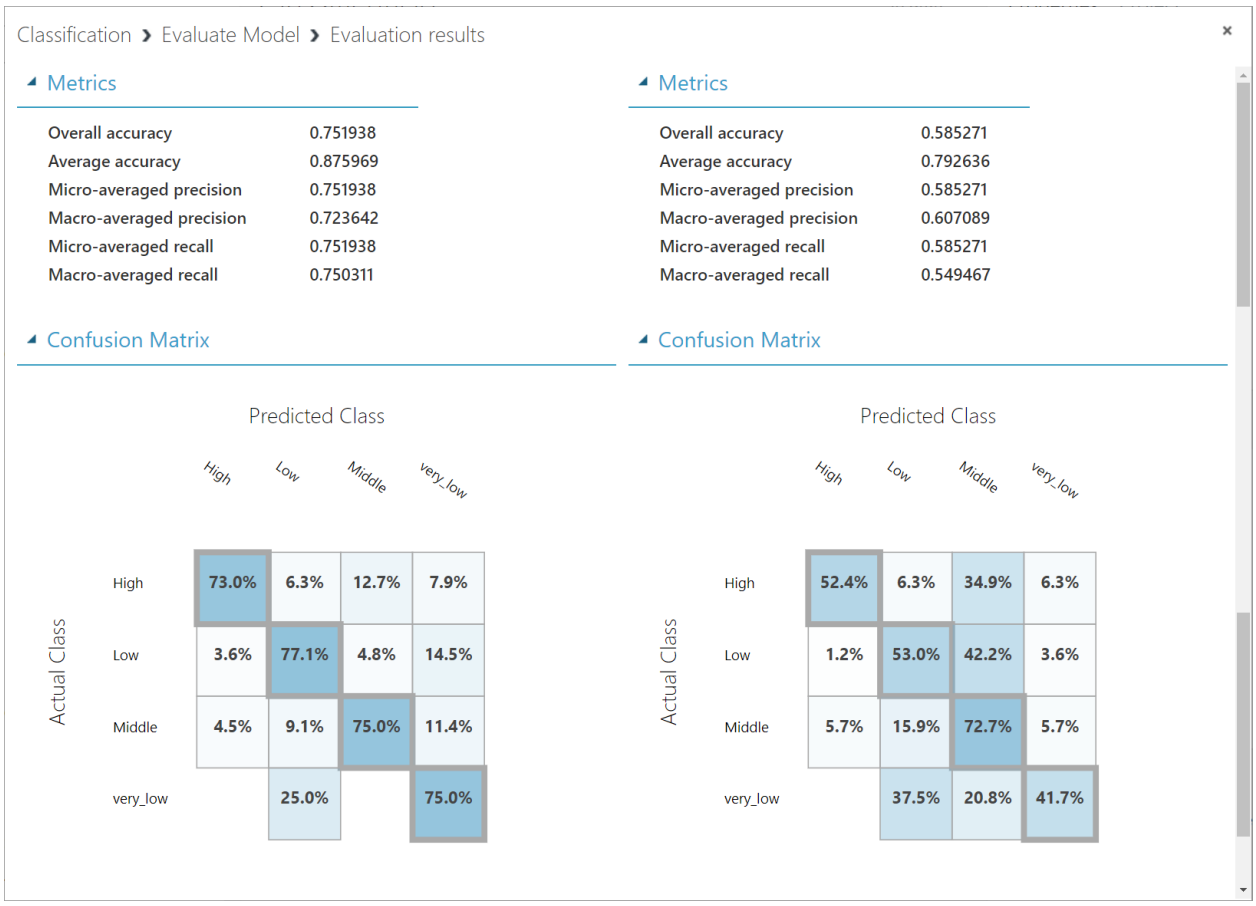


Шаг 8:

1) Сравнить две получившиеся модели. Перетащить объект Evaluate Model в область эксперимента: Search.. => Evaluate Model;

2) Визуализировать данные. Выбрать Visualize.

Визуализация результатов:



1) Overall accuracy – точность – измеряет качество модели классификации как отношение истинных результатов к общему количеству
2) случаев.
3) Average accuracy – это соотношение истинных результатов ко всем правильным результатам, которая вернула модель.
4) F-мера – оценка рассчитывается как средневзвешенное значение точности между 0 и 1 (или в процентном соотношении), где идеальное значение равно 1.
5) AUC измеряет площадь под кривой, построенной с истинными положительными значениями на оси Y и ложными положительными значениями на оси X. Эта метрика полезна, потому что она предоставляет одно число, которое позволяет сравнивать модели разных типов.

2.2. Microsoft Cognitive Services

Код программной реализации решения задачи представлен ниже:

```
import asyncio
import io
import glob
import os
import sys
import time
import uuid
import requests
from urllib.parse import urlparse
from io import BytesIO
from PIL import Image, ImageDraw, ImageFont
from azure.cognitiveservices.vision.face import FaceClient
from msrest.authentication import CognitiveServicesCredentials
import json
from azure.cognitiveservices.vision.face.models import TrainingStatusType, Person, SnapshotObjectType, \
    OperationStatusType

import cognitive_face as CF

FACE_SUBSCRIPTION_KEY = "e9fa3a8292fc470c85db68cadc806961"
FACE_SUBSCRIPTION_KEY = "b0c394ce0c8d43219876638135bc8cbb"
FACE_ENDPOINT = "https://westcentralus.api.cognitive.microsoft.com/face/v1.0/detect"
FACE_ENDPOINT_LIB = "https://westcentralus.api.cognitive.microsoft.com/face/v1.0/"
COMPUTER_VISION_ENDPOINT = "https://westcentralus.api.cognitive.microsoft.com/"

# Set the FACE_SUBSCRIPTION_KEY environment variable with your key as the value.
# This key will serve all examples in this document.
os.environ['FACE_SUBSCRIPTION_KEY'] = FACE_SUBSCRIPTION_KEY
KEY = os.environ['FACE_SUBSCRIPTION_KEY']

# Set the FACE_ENDPOINT environment variable with the endpoint from your Face service in Azure.
# This endpoint will be used in all examples in this quickstart.
os.environ['COMPUTER_VISION_ENDPOINT'] = COMPUTER_VISION_ENDPOINT
os.environ['FACE_ENDPOINT'] = FACE_ENDPOINT
ENDPOINT = os.environ['COMPUTER_VISION_ENDPOINT']

# Create an authenticated FaceClient.
face_client = FaceClient(ENDPOINT, CognitiveServicesCredentials(KEY))

### Определение лиц на изображении ###
# Detect a face in an image that contains a single face
single_face_image_url = 'https://www.biography.com/.image/t_share/MTQ1MzAyNzYzOTgxNTE0NTEz/john-f-kennedy---mini-biography.jp
single_image_name = os.path.basename(single_face_image_url)
detected_faces = face_client.face.detect_with_url(url=single_face_image_url)
if not detected_faces:
    raise Exception('No face detected from image {}'.format(single_image_name))

# Display the detected face ID in the first single-face image.
# Face IDs are used for comparison to faces (their IDs) detected in other images.
print('Detected face ID from', single_image_name, ':')
for face in detected_faces: print(face.face_id)
print()

# Save this ID for use in Find Similar
first_image_face_ID = detected_faces[0].face_id
```

```python
### Отображение лиц и их выделение рамкой ###
# Detect a face in an image that contains a single face
single_face_image_url = 'https://i.ibb.co/RTnn06h/image.jpg'
single_image_name = os.path.basename(single_face_image_url)
params = {
                'returnFaceId': 'true',
                'returnFaceLandmarks': 'false',
                'returnRectangle': 'true',
                'returnFaceAttributes': 'age,gender,headPose,smile,facialHair,glasses,emotion,hair,makeup,occlusion,'
                                        'accessories,blur,exposure,noise',
            }
headers = {'Ocp-Apim-Subscription-Key': FACE_SUBSCRIPTION_KEY}
response = requests.post(FACE_ENDPOINT, params=params,
                         headers=headers, json={"url": single_face_image_url})
detected_faces = response.json()

if not detected_faces:
    raise Exception('No face detected from image {}'.format(single_image_name))


# Convert width height to a point in a rectangle
def getRectangle(faceDictionary):
    rect = faceDictionary['faceRectangle']
    left = rect['left']
    top = rect['top']
    right = left + rect['width']
    bottom = top + rect['height']
    return ((left, top), (right, bottom))
# Download the image from the url
response = requests.get(single_face_image_url)
img = Image.open(BytesIO(response.content))

# For each face returned use the face rectangle and draw a red box.
print('Drawing rectangle around face... see popup for results.')
draw = ImageDraw.Draw(img)
for n, face in enumerate(detected_faces):
    draw.rectangle(getRectangle(face), outline='red')
    text = 'age: '+str(face['faceAttributes']['age'])
    font = ImageFont.truetype("arial.ttf", 22, encoding='UTF-8')
    w, h = font.getsize(text)
    x, y = getRectangle(face)[0][0], getRectangle(face)[1][1]
    draw.rectangle((x, y, x + w, y + h), fill='black')
    draw.text((x, y), text,fill="red", font=font, stroke_fill='green')

    print('человек номер - ', n)
    face_attributes = face['faceAttributes']
    print("пол: ", face_attributes['gender'])
    print("возраст: ", face_attributes['age'])
    print("есть ли очки?: ", face_attributes['glasses'])
    print('Эмоция: ')
    for key, value in face_attributes['emotion'].items():
        print('{}: {}'.format(key, value))

# Display the image in the users default image browser.
img.show()
```

Результат работы программы:

```
Drawing rectangle around face... see popup for results.
Similar faces found in president-family-portrait-closeup.jpg:
пол:  male
возраст:  56.0
есть ли очки?:  NoGlasses
Эмоция:
anger: 0.001
contempt: 0.001
disgust: 0.0
fear: 0.0
happiness: 0.0
neutral: 0.998
sadness: 0.0
surprise: 0.0

Process finished with exit code 0
```

3. Список источников

1) Материалы портала URL: https://studio.azureml.net/ - дата обращения 24.04.2020.
2) Документация по Microsoft Machine Learning Studio URL: https://drive.google.com/file/d/1ZFrdmAXWOZseJM0qL-u2P6OYn365vZVg/view?usp=sharing – дата обращения 24.04.2020.
3) Материалы портала URL: https://docs.microsoft.com/en-us/azure/cognitive-services/face/quickstarts/python – дата обращения 05.05.2020.
4) Книга по построению приложений с использованием Microsoft Cognitive Services URL: https://drive.google.com/file/d/1KM4gcxuMURk4u--NqPLCgHzmHBskijCP/view?usp=sharing - дата обращения 24.04.2020.
5) Материалы портала URL: https://westus.dev.cognitive.microsoft.com/docs/services/563879b61984550e40cbbe8d/operations/563879b61984550f30395236 - дата обращения 05.05.2020.