Лабораторная работа № 1
РАЗРАБОТКА ПРОЕКТА ПО КЛАССИФИКАЦИИ РУКОПИСНЫХ ЦИФР НА ПЛАТФОРМЕ TEACHABLE MACHINE

Время выполнения – 8 часов (аудиторная работа – 2 часа, самостоятельная работа – 6 часов).

Цель работы: освоить алгоритм классификации изображений с использованием платформы *Teachable Machine* и датафреймов, размещенных на сайте *Kaggle*.

Задачи работы

1. Изучить возможности платформы визуального программирования *Teachable Machine*.
2. Научиться находить готовые наборы данных для машинного обучения на платформе *Kaggle*.
3. Освоить алгоритм классификации изображений с использованием *Teachable Machine* и готовых датасетов *Kaggle*.

Перечень обеспечивающих средств

1. Платформа *Kaggle* для поиска и загрузки наборов данных для машинного обучения.
2. Платформа визуального программирования *Teachable Machine*.
3. Персональный компьютер с доступом в интернет.
4. Современный веб-браузер для работы с платформами (например, *Google Chrome*).

Общие теоретические сведения

Kaggle – это открытая платформа для работы с данными и машинным обучением. На платформе доступны готовые наборы данных, инструменты для анализа, а также возможность взаимодействия с другими специалистами.

Teachable Machine – это онлайн-инструмент для создания моделей машинного обучения, предназначенных для распознавания изображений, звуков и поз человека. Основное преимущество платформы заключается в её доступности: для работы с инструментом не требуется программирование, а процесс построения модели осуществляется через интуитивно понятный интерфейс.

1. Регистрация на *Kaggle* и подготовка данных.

Зарегистрируйтесь на платформе *Kaggle* (https://www.kaggle.com).

Перейдите в раздел *Datasets* / *Classification* и найдите набор данных «Handwritten Digits 0–9» (рис. 1) или воспользуйтесь прямой ссылкой: https://www.kaggle.com/datasets/olafkrastovski/handwritten-digits-0-9.

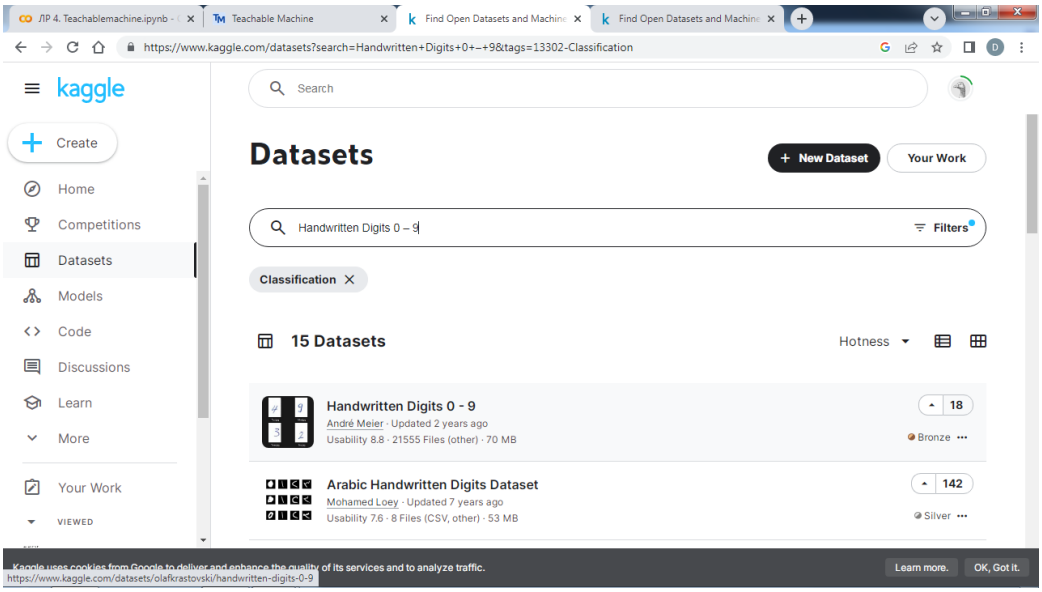

Рис. 1. Платформа *Kaggle*

Скачайте архив набора данных на локальный диск и разархивируйте его. В случае невозможности скачать набор данных с сайта Kaggle.com вы можете загрузить его по ссылке: https://drive.google.com/file/d/15nY2-p-iSZQPWe-9-cEslNJD9pzx_axR/view?usp=drive_link.

После этого в вашей рабочей папке появятся 10 папок, каждая из которых содержит изображения одной из цифр от 0 до 9.

2. Создание модели на платформе *Teachable Machine*.

Перейдите на платформу Teachable Machine (https://teachablemachine.withgoogle.com).

Нажмите кнопку «Начать» и выберите «Новый проект с изображениями / Стандартная модель изображения».

Создайте классы для каждой цифры (0–9) и загрузите в них соответствующие изображения из набора данных Kaggle.

3. Обучение модели.

В разделе дополнительных параметров обучения обратите внимание на параметр «Эпохи». Эпоха – это один полный проход всех данных через модель. Увеличение числа эпох обычно улучшает качество модели, но может занять больше времени.

Нажмите «Обучить», чтобы запустить процесс обучения модели (рис. 2).

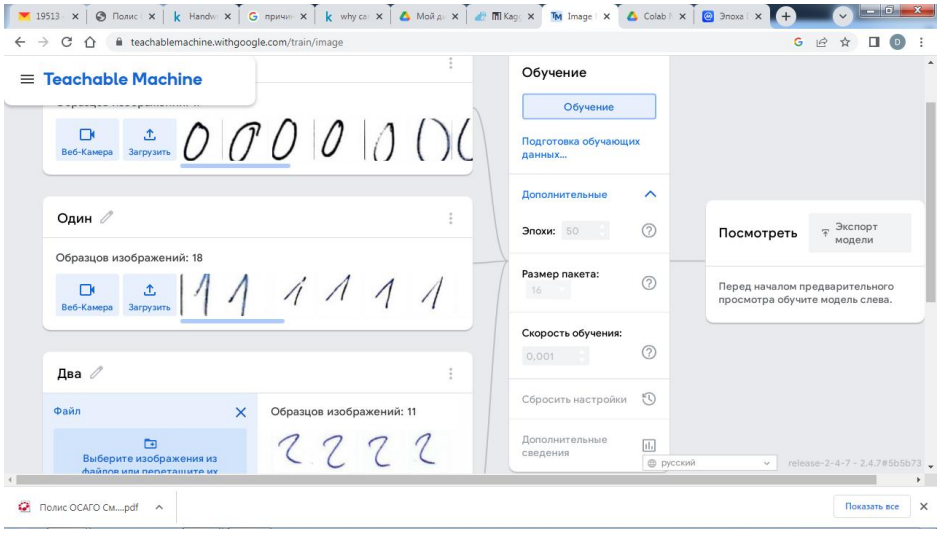

Рис. 2. Обучение модели по распознаванию изображений рукописных цифр

4. Тестирование модели.

Выберите любое изображение цифры из набора данных и откройте его в графическом редакторе (например, *Paint*).

Удалите изображение цифры ластиком и нарисуйте новую цифру, затем сохраните файл в формате *JPG* (рис. 3).

Загрузите созданное изображение в обученную модель.

Оцените точность распознавания новой цифры моделью (рис. 4).

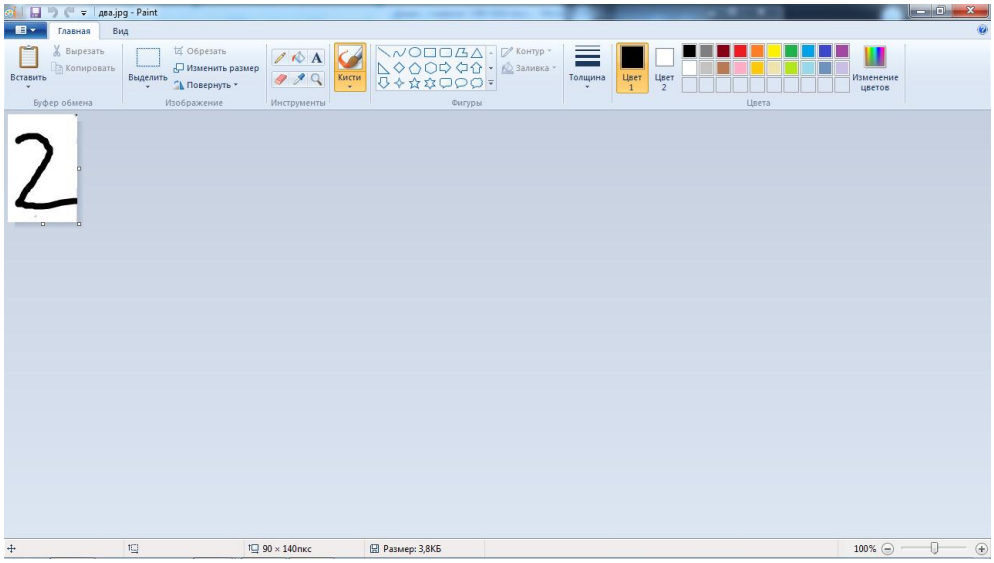

Рис. 3. Изображение рукописной цифры

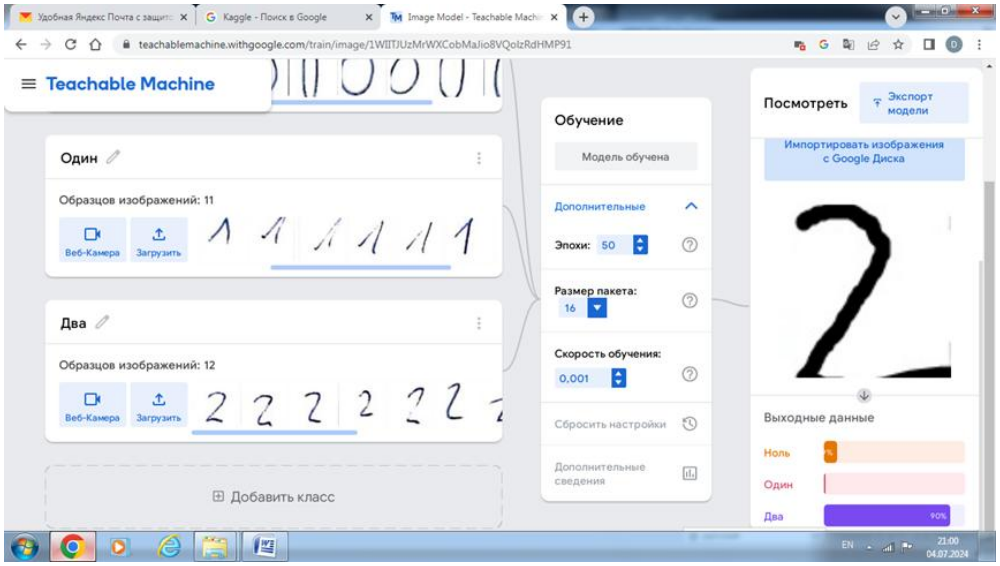

Рис. 4. Точность распознавания нарисованной цифры

5. Анализ результатов.

Откройте раздел «Дополнительные сведения» в *Teachable Machine* и выберите опцию «Вычислить точность на каждый класс» (рис. 5).

Программа автоматически разделит загруженные изображения каждого класса на две выборки: обучающую и тестируемую.

Обучающая выборка (80 % изображений) используется для обучения модели.

Тестируемая выборка (20 % изображений) применяется для проверки её точности.

Во время обучения нейросеть принимает на вход изображения из обучающей выборки, а на выходе – соответствующие им названия классов (метки). Модель формирует взаимосвязь между входными изображениями и их классами.

В процессе тестирования нейросети на вход подаются изображения из тестируемой выборки, но метки классов при этом не предоставляются. Нейросеть самостоятельно предсказывает класс для каждого изображения.

Предсказанные моделью классы сравниваются с фактическими метками из тестируемой выборки.

Ключевые метрики:

– SAMPLES – количество изображений, использованных для тестирования;

– ACCURACY – доля изображений, для которых модель правильно определила класс.

Этот процесс позволяет оценить, насколько точно обученная модель справляется с классификацией изображений из ранее незнакомой выборки.

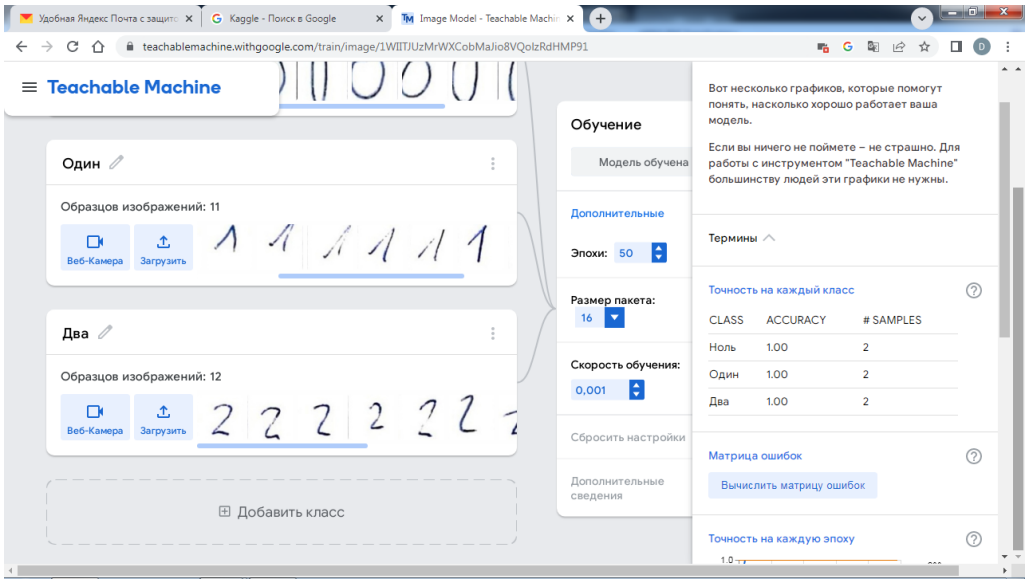

Рис. 5. Точность обученной модели

6. Сохранение проекта.

Сохраните проект на *Google Диск* с именем по шаблону: Фамилия_студента Название_группы (например, Иванов БИ-23).

Для сохранения откройте меню (символ ≡) и выберите «Сохранить проект на Диск».

Задания

В заданиях используется условное обозначение: N – порядковый номер первой буквы фамилии студента согласно русскому алфавиту (например, А = 1, Б = 2 и т. д.).

Задание 1. Распознавание рукописных цифр.

1. Обучите модель на платформе *Teachable Machine* для распознавания рукописных цифр, используя набор данных «*Handwritten Digits* 0 – 9».

2. Установите количество эпох, равное $N + 50$.

3. После обучения распознайте две рукописные цифры, соответствующие порядковому номеру первой буквы фамилии студента (например, для буквы «А» это 0 и 1).

	А	Б	В	Г	Д	Е	Ё	Ж	З
Цифры	0	0	0	0	0	0	0	0	0
	1	2	3	4	5	6	7	8	9
	И	Й	К	Л	М	Н	О	П	Р
Цифры	1	1	1	1	1	1	1	1	1
	2	2	3	4	5	6	7	8	9
	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ
Цифры	2	2	2	2	2	2	2	3	3
	3	4	5	6	7	8	9	4	5
	Ъ	Ы	Ь	Э	Ю	Я			
Цифры	3	3	3	3	4	4			
	6	7	8	9	5	6			

Задание 2. Распознавание арифметических знаков.

1. Скачайте с сайта *Kaggle* набор данных «*Basic arithmetics dataset*» (https://www.kaggle.com/datasets/simepavlic/basic-arithmetics-dataset или https://drive.google.com/file/d/1xdJxtH6ZfPKfjMnvOHGfBM1BtOx5UBdZ/view?usp=sharing).

2. Создайте на платформе *Teachable Machine* новый проект по распознаванию следующих арифметических знаков:

– division (деление);

– left_bracket (левая скобка);

– minus (минус);

– plus (плюс);

– right_bracket (правая скобка);

– multiplication (умножение).

3. Установите количество эпох, равное 70 – *N*.

4. После обучения распознайте арифметический знак, соответствующий порядковому номеру первой буквы фамилии студента (например, для буквы «А» это символ деления).

Знак	: деле-ние	(левая скобка	- минус	+ плюс	) правая скобка	* умножение
Буква	А	Б	В	Г	Д	Е
	Ё	Ж	З	И	К	Л
	М	Н	О	П	Р	С
	Т	У	Ф	Х	Ц	Ч
	Ш	Щ	Ы	Э	Ю	Я

Контрольные вопросы

1. Какие этапы включают процесс обучения модели в *Teachable Machine*?

2. Что такое эпоха в контексте машинного обучения?

3. Как параметры выборок (обучающей и тестирующей) влияют на качество модели?

4. Какие преимущества и ограничения имеет использование *Teachable Machine* для решения задач машинного обучения?

Содержание отчета

1. Титульный лист
2. Цель работы.
3. Формулировка задания.
4. Описание результатов выполнения задания в текстовом и графическом виде (скриншоты).
5. Ответы на контрольные вопросы.
6. Подробный вывод о проделанной работе.

Лабораторная работа № 2
ПОСТРОЕНИЕ АЛГОРИТМА КЛАССИФИКАЦИИ ЦВЕТОВ ИРИСА С ПОМОЩЬЮ ВИДЖЕТОВ НА ПЛАТФОРМЕ ORANGE DATA MINING

Время выполнения – 8 часов (аудиторная работа – 2 часа, самостоятельная работа – 6 часов).

Цель работы: изучить механизм решения задачи классификации объектов с использованием технологий искусственного интеллекта на платформе *Orange Data Mining*.

Задачи работы

1. Освоить навыки работы со специализированной платформой визуального программирования *Orange Data Mining*.

2. Ознакомиться со структурой датасетов, применяемых в задачах машинного обучения.

3. Изучить этапы разработки и обучения модели классификации объектов с помощью виджетов на платформе.

Перечень обеспечивающих средств

1. Программное обеспечение *Orange Data Mining*.

2. Датасет для работы *Iris Dataset*, предустановленный в платформе.

Общие теоретические сведения

Orange Data Mining – это платформа для визуализации данных, процессов машинного обучения и интеллектуального анализа данных.

Основными элементами платформы являются виджеты, которые выполняют отдельные функции:

– подготовка данных для машинного обучения;

– визуализация (отображение) этих данных в виде удобных для восприятия таблиц;

– деление данных на обучающую и тестируемую выборки;

– выбор алгоритма обучения модели;

– обучение модели на основе выбранного алгоритма;

– определение качества модели.

Создание процесса машинного обучения осуществляется за счёт соединения виджетов, где входные и выходные сигналы определяют передачу данных между ними.

Рассмотрим основные этапы работы в *Orange Data Mining*.

1. Установка Orange Data Mining.

Перейдите на сайт *Orange Data Mining* (https://orangedatamining.com/download/). Скачайте и установите последнюю версию программы для вашей операционной системы. После установки запустите программу.

2. Работа с набором данных *Iris*.

Создайте новый рабочий процесс через виджет *New Workflow*.

Для загрузки набора данных можно воспользоваться несколькими виджетами.

Виджет *File* считывает данные из файла или загружает по *URL*.

Виджет *Datasets* извлекает выбранный набор данных с сервера, загружает его в локальную память и становится доступен без подключения к Интернету. Каждый из доступных набором данных снабжен описанием (количество объектов в наборе, количестве характеристик объектов и т. д.).

Перетащите виджет *Datasets* на холст.

Дважды щёлкните на виджет *Datasets* и выберите набор данных *Iris*.

Набор данных *Iris* предназначен для решения задачи классификации объектов с использованием технологии машинного обучения. Набор данных о цветках ириса содержит информацию о 150 цветках. Каждый цветок характеризуется 5 характеристиками, включая класс (метку), к которому цветок относится. Все цветки ириса подразделяются на три класса.

Каждый виджет имеет входные и выходные сигналы. Сигнал определяет данные, которые поступают на вход виджету или являются его результатом. При получении входного сигнала виджет выполняет определенные действия и отправляет соответствующие сигналы связанным с ним виджетам.

Виджет *Datasets* имеет выходной сигнал *Data*, который поступает в качестве входного сигнала на виджет *Data Table*.

3. Отображение данных в таблице.

Виджет *Data Table* выводит данные из файла на экран в виде удобной для восприятия таблицы. Перетащите виджет *Data Table* на холст.

Соедините виджеты *Datasets* и *Data Table* (сигнал *Data*). При создании связи между виджетами входной и выходной сигналы выбираются автоматически. При обновлении файла *Datasets* обновляется *Data Table*. Связь между двумя виджетами подписывается над стрелкой (рис. 1).

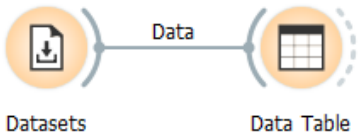

Рис. 1. Установление связи между виджетами

Дважды щёлкните на *Data Table*, чтобы просмотреть содержимое набора данных.

4. Визуализация данных.

Раздел *Visualization* объединяет виджеты, используемые для представления наборов данных в виде графиков:

– виджет *Box Plot* показывает медиану, нижний и верхний квантили, минимальное и максимальное значение выборки и выбросы;

– виджет *Distributions* для построения диаграммы частотного распределения признака;

– виджет *Heat Map* для построения тепловой диаграммы;

– виджет *Venn Diagram* для построения диаграммы Венна;

– виджет *Sieve Diagram* для построения диаграммы Ридвиля и Шюпбаха;

– виджет *Pythagorean Tree* и *Pythagorean Forest* для построения деревьев Пифагора;

– виджет *Mosaic Display* для построения мозаичной диаграммы;

– виджет *Tree Viewer* для визуального представления древовидных структур;

– виджет *FreeViz* и *Radviz* для визуализации многомерных данных.

Виджет *Scatter Plot* позволяет строить двумерные графики по выбранным признакам.

Виджет *Scatter Plot* имеет три входных сигнала:

– связь *Data* используется для отображения на графике всех данных;

– связь *Data Subset* используется для отображения на графике подмножества данных;

– связь *Features*.

Перетащите виджет *Scatter Plot* на холст.

Соедините его с *Data Table* (сигнал *Data*). В качестве осей выберите признаки ириса: *petal length* (длина лепестка) и *sepal length* (длина чашелистика).

Постройте график распределения данных (рис. 2).

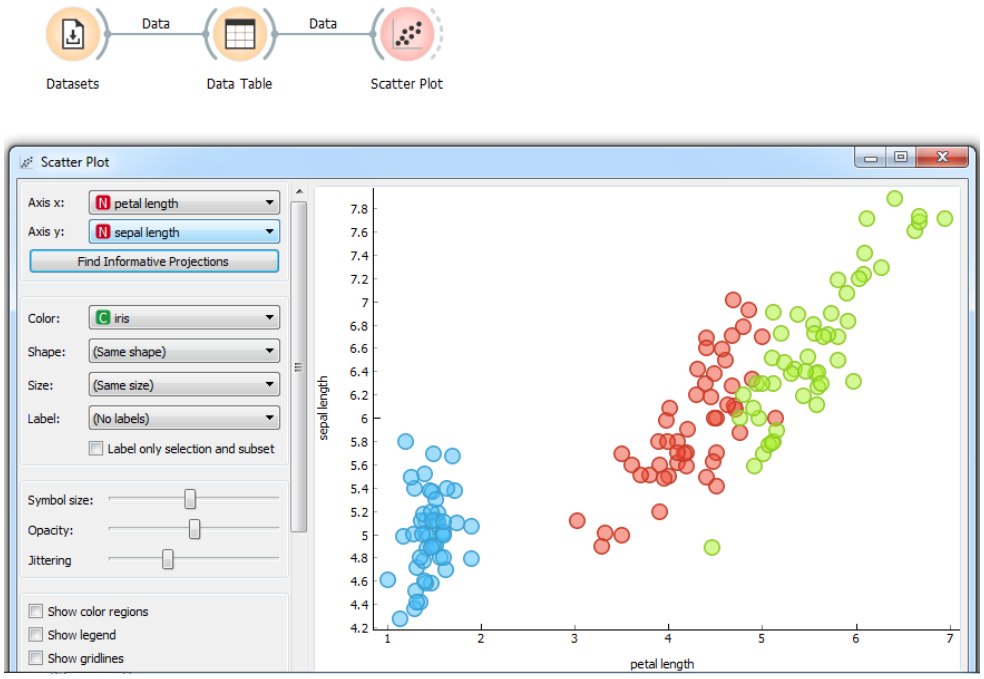

Рис. 2. Представление набора данных в виде графика

5. Обучение моделей классификации.

Раздел *Model* объединяет виджеты алгоритмов, с помощью которых осуществляется обучение модели (рис. 3).

Перетащите на холст три виджета: *kNN* (метод *k*-ближайших соседей), *Logistic Regression* (метод логистической регрессии), *Random Forest* (метод случайных деревьев).

Самый простой из перечисленных методов классификации – это метод *k*-ближайших соседей. В соответствие с данным методом объект (цветок ириса) относится к тому классу, который наиболее распространён среди *k*-соседей данного объекта, классы которых уже известны.

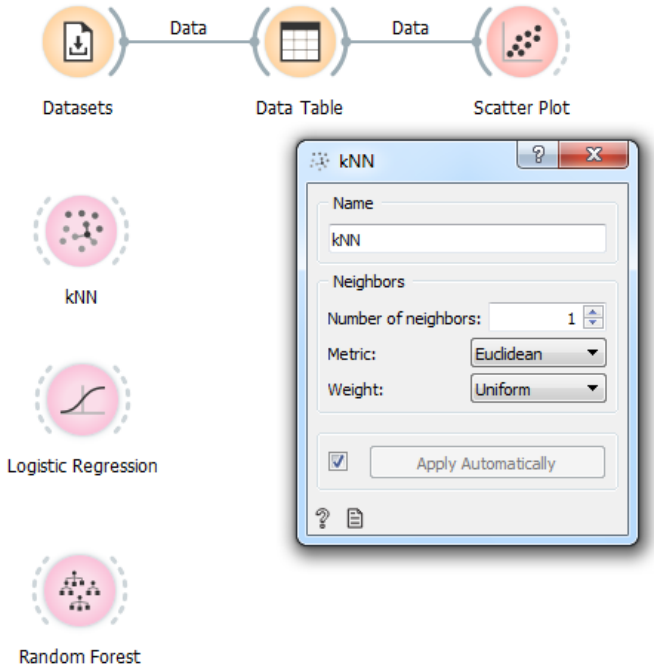

Рис. 3. Методы классификации объектов

На рис. 4 новый объект (зелёный круг) классифицируется как красный треугольник, если $k = 3$, или как синий квадрат, если $k = 5$.

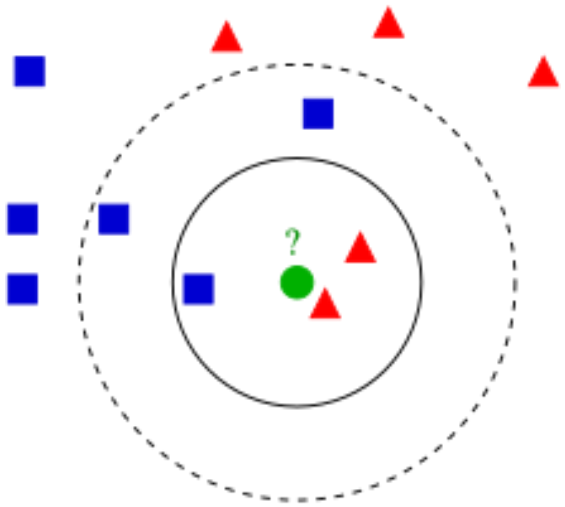

Рис. 4. Метод *k*-ближайших соседей

Соедините виджет *Datasets* с этими виджетами (сигнал *Data*).

6. Оценка качества моделей.

Раздел *Evaluate* объединяет виджеты для тестирования качества алгоритмов построения модели.

Для тестирования создаваемой модели воспользуйтесь виджетом *Test and Score*.

Виджет *Test and Score* принимает следующие входные сигналы:

- связь *Data* соединяет виджет *Test and Score* с виджетом *Datasets* для загрузки данных, на которых будет обучена модель;

- связь *Test Data* соединяет виджет *Test and Score* с виджетом *Datasets* для загрузки данных для проверки модели;

- связь *Learner* соединяет виджет *Test and Score* с виджетами алгоритмов обучения модели.

Виджет *Test and Score* позволяет тестировать классификаторы несколькими способами.

1. *k-Fold Cross Validation* (перекрестная проверка на *k* группах). Набор данных перемешивается случайным образом и подразделяется на *k* групп. Состав групп в течение всей процедуры кроссвалидации остается неизменным. Одна группа используется для тестирования модели, остальные группы ($k - 1$) – для

обучения модели с помощью выбранного алгоритма классификации. Для каждого из сравниваемых алгоритма классификации тестирующая и обучающие группы подбираются случайным образом.

2. *Random sampling* (случайная разбивка в заданном соотношении). Набор данных подразделяется случайным образом на обучающую и тестируемую выборки в заданном соотношении. На обучающей выборке осуществляется обучение модели, на тестируемой выборке – проверка надежности модели. Тестирование повторяется заданное количество раз, результаты тестирований усредняются.

Перетащите виджет *Test and Score* на холст.

Соедините виджеты алгоритмов классификации (*kNN*, *Logistic Regression*, *Random Forest*) с виджетом *Datasets* связью *Data*, а с виджетом *Test and Score* связью *Learner*. Соедините виджет *Datasets* с виджетом *Test and Score* связью *Data*.

В качестве способа проверки в параметрах виджета *Test and Score* выберите способ *Random sampling*.

Размер обучающей выборки установите равным 80 %, количество тестирований равным 10 (рис. 5).

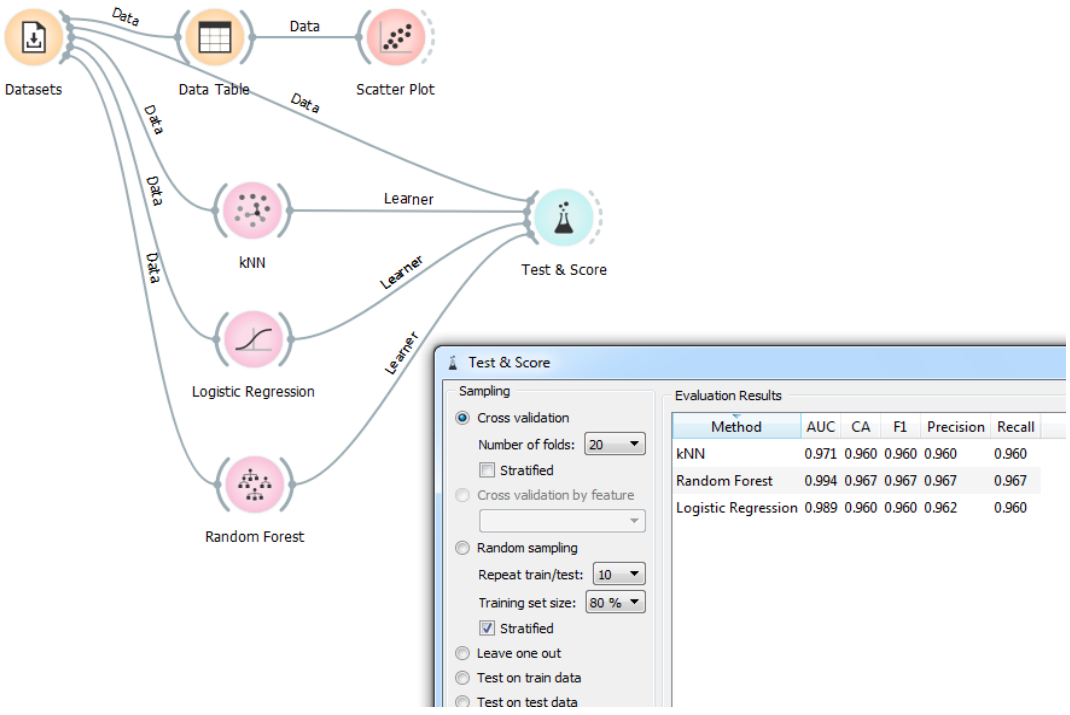

Рис. 5. Оценка надежности алгоритмов классификации

Сравните алгоритмы по показателям:

– *AUC* (соотношение между долей правильно классифицированных объектов и долей ошибочно классифицированных объектов);

– *Classification accuracy* (отношение количества правильно классифицированных объектов к общему количеству объектов);

– *F*1 (соотношение между показателями точности *Precision* и полноты *Recall*, вычисляемое по формуле

*F*1 = 2*(*Precision***Recall*)/(*Precision*+*Recall*),

где *Precision* – доля объектов, фактически принадлежащих к данному классу, по отношению ко всем объектам, которые модель классифицировала в качестве принадлежащих к этому классу;

Recall – доля объектов, которые модель классифицировала в качестве принадлежащих к этому классу, по отношению ко всем объектам, фактически принадлежащих к данному классу.

7. Построение матрицы несоответствий.

Для большей наглядности результатов сравнения алгоритмов классификации можно использовать виджет *Confusion Matrix*.

Добавьте на холст виджет *Confusion Matrix*, соедините его с *Test and Score* и постройте матрицу несоответствий для каждого алгоритма (рис. 6).

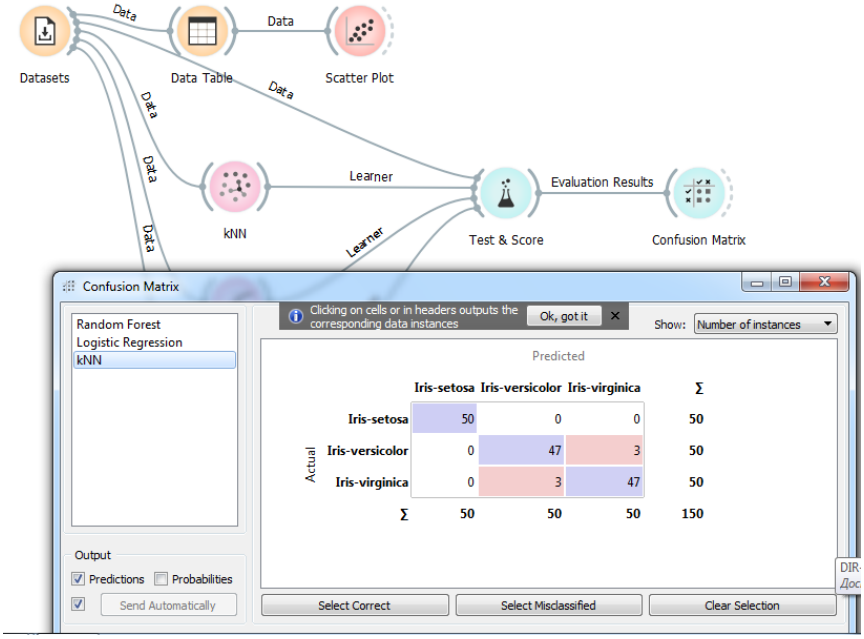

Рис. 6. Матрица несоответствий

8. Создание и модификация нового набора данных.

Создайте новый набор данных из цветков ириса на основе датасета *Iris*. Для этого перетащите виджет *Data Save* на холст, соедините его с виджетом *Datasets* с помощью связи *Data* и сохраните на локальном диске.

Откройте сохраненный файл с помощью текстового редактора и оставьте в файле количество строк, соответствующее номеру варианта + 10. Остальные строки удалите. Удалите также названия классов объектов. Данные в строках немного измените (рис. 7).

Сохраните новый набор данных на локальном диске.

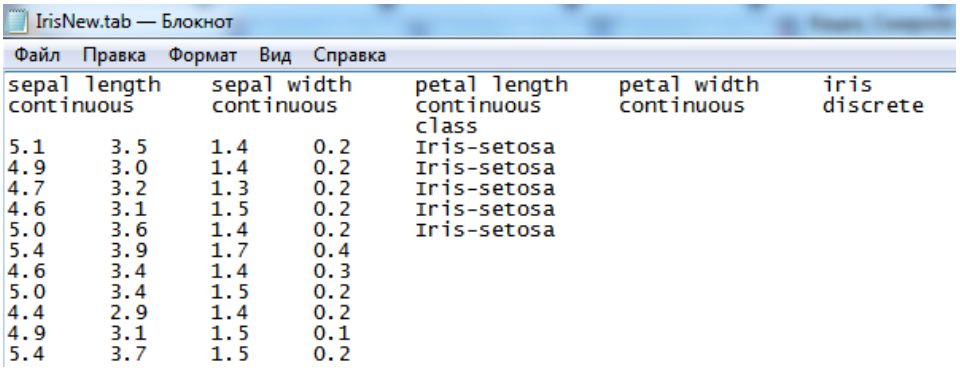

Рис. 7. Сохранение набора данных Iris

9. Загрузка и использование нового набора данных.

Перетащите на холст виджет *File* и загрузите новый набор данных.

Перетащите на холст виджет *Data Table* и соедините его с виджетом *File*. Выведите на экран новый набор данных (рис. 8).

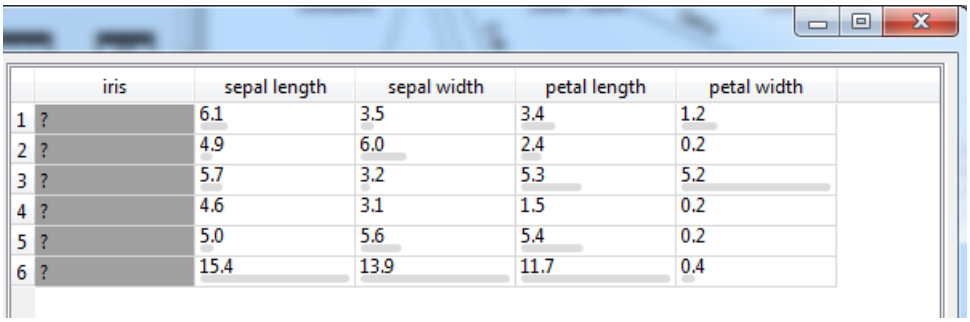

Рис. 8. Табличная форма нового набора данных

10. Классификация новых объектов.

Перетащите на холст виджет *Prediction* из раздела *Evaluate*. Соедините виджет *Prediction* с виджетом *Data Table* (1) связью *Data*. Соедините виджет *Prediction* с виджетами *kNN*, *Logistic Regression*, *Random Forest*. Посмотрите результаты классификации новых объектов (рис. 9).

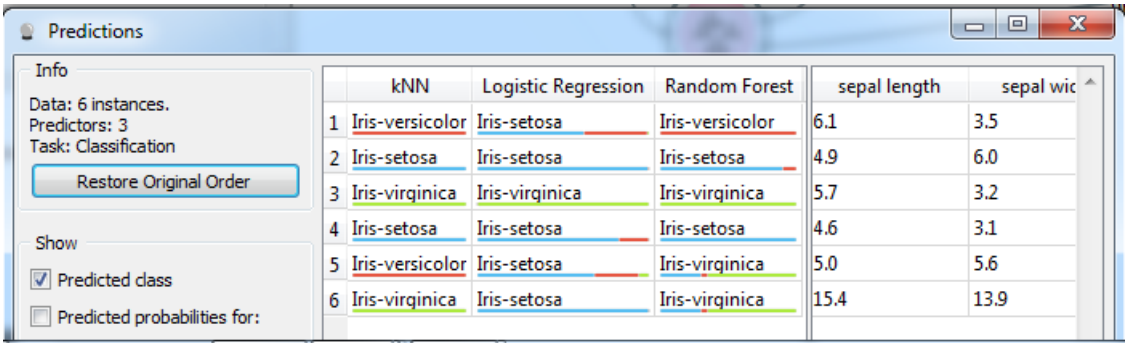

Рис. 9. Результаты классификации новых объектов

11. Сохранение рабочего процесса.

Сохраните созданный рабочий процесс на локальный диск через меню *File / Save Workflow*.

Задания

В заданиях используются условные обозначения:

– *N* – порядковый номер первой буквы фамилии студента согласно русскому алфавиту;

– *M* – порядковый номер первой буквы имени студента согласно русскому алфавиту.

Задание 1. Работа с набором данных *Iris*.

В разработанном с помощью виджетов рабочем процессе:

– создайте новый набор данных из цветков ириса на основе датасета *Iris*;

– количество объектов (цветков ириса) в новом наборе данных должно равняться *N*;

– в параметрах метода *kNN* установите количество ближайших соседей равным *M*.

– проведите классификацию новых объектов (цветков ириса).

Задание 2. Работа с набором данных *Wine*.

Создайте новый рабочий процесс с набором данных *Wine*:

– создайте новый набор данных из бутылок вина на основе датасета *Wine*;

– количество объектов (бутылок вина) в новом наборе данных должно равняться *N*;

– в параметрах метода *kNN* установите количество ближайших соседей равным *M*;

– проведите классификацию новых объектов (бутылок вина).

Контрольные вопросы

1. Что представляет собой платформа *Orange Data Mining*?
2. Перечислите основные разделы виджетов, доступные на платформе.
3. Какие виджеты используются для загрузки наборов данных?
4. Как подключить виджет для визуализации данных?
5. Какие виджеты можно использовать для разделения данных на выборки?
6. Для чего используется виджет *Scatter Plot*?
7. Перечислите другие виджеты из раздела *Visualization*, которые можно использовать для анализа набора данных.
8. Как работает метод *k*-ближайших соседей?
9. Чем отличаются методы *Logistic Regression* и *Random Forest*?
10. Какие параметры можно настроить в виджете *Test and Score*?
11. Какие методы оценки качества модели доступны в *Orange*?
12. Что такое метрики AUC, Classification Accuracy и F1, и как их интерпретировать?

Содержание отчета

1. Титульный лист
2. Цель работы.
3. Формулировка задания.

4. Описание результатов выполнения задания в текстовом и графическом виде (скриншоты).

5. Ответы на контрольные вопросы.

6. Подробный вывод о проделанной работе.

Лабораторная работа № 3
РАЗРАБОТКА МОДЕЛИ МАШИННОГО ОБУЧЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КЛАССИФИКАЦИИ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕК, МОДУЛЕЙ И ФРЕЙМВОРКОВ PYTHON

Время выполнения – 8 часов (аудиторная работа – 2 часа, самостоятельная работа – 6 часов).

Цель работы: решение задачи классификации объектов на примере цветков ириса и бутылок вина с помощью программного кода, написанного на языке *Python* в блокнотах на платформе *Google Colab*.

Задачи работы

1. Изучить библиотеки, модули и фреймворки *Python*, применяемые для машинного обучения.
2. Освоить навыки работы на платформе *Google Colab*.
3. Научиться решать задачи классификации объектов с использованием программного кода на языке *Python*.

Перечень обеспечивающих средств

1. Платформа *Google Colab*.
2. Библиотеки, модули и фреймворки *Python* для машинного обучения.

Общие теоретические сведения

Jupyter Notebook – это среда для разработки и выполнения программного кода и его отдельных фрагментов. Она работает офлайн.

Google Colab (далее *Colab*) – облачный сервис, который позволяет использовать файлы *Jupyter Notebook* без их установки на локальный компьютер.

Файлы, созданные в *Jupyter Notebook* или *Colab*, имеют расширение *.ipynb* и называются блокнотами (тетрадями, ноутбуками). В них можно писать код на

различных языках программирования, однако для задач искусственного интеллекта чаще всего используется *Python*. Популярность *Python* обусловлена наличием множества библиотек и фреймворков для работы с искусственным интеллектом и машинным обучением.

Блокноты *Colab* хранятся в облаке *Google* Диск, в папке *Colab Notebooks*, которая создаётся автоматически. Для работы в *Colab* требуется аккаунт *Google*.

Рассмотрим подробно решение задачи классификации ирисов с использованием машинного обучения и программного кода на языке *Python*. Создаваемая модель будет копией модели, разработанной в рамках предыдущей лабораторной работы и реализованной с помощью виджетов на платформе *Orange Data Mining*.

Для создания модели потребуются модули, фреймворки и датасеты библиотеки *scikit-learn* (версия 1.2.2).

1. Работа с блокнотом *Colab*.

Перейдите на сайт *Google Colab* (https://colab.research.google.com/) и выберите опцию *New Notebook* (Создать новый блокнот).

Войдите в аккаунт *Google*.

В блокноте информация размещается в ячейках, которые бывают двух типов: *кодовые* и *текстовые*. Текстовые ячейки используются для написания пояснений к программному коду, отображения изображений, видео и т. д. Кодовые ячейки используются для написания и выполнения программного кода.

Чтобы добавить кодовую или текстовую ячейку в *Colab*, нужно выбрать соответствующий пункт в меню Вставка или подвести указатель мыши к середине нижней границы ячейки и выбрать одну из кнопок: «+ Код» или «+ Текст».

В блокноте создайте текстовую ячейку, в которой напишите текст без кавычек: «Лабораторная работа студента группы … Фамилия, имя», указав свои данные: номер группы, фамилию и имя.

Далее мы будем создавать кодовые ячейки, в которые будем вводить программный код. Также в кодовых ячейках мы будем указывать комментарии к программному коду. Они начинаются с символа #.

После ввода программного кода необходимо его запускать для получения результата.

Установите пакет *scikit-learn*. Если библиотека уже установлена, этот шаг можно пропустить.

```
# загружаем библиотеку scikit-learn
!pip install scikit-learn
```

2. Установка и импорт библиотек.

Установите библиотеку *sklearn*, а также различные модули, необходимые для работы.

```
# импортируем библиотеки и модули
import sklearn
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

Загрузите набор данных *load_iris* из модуля *datasets* библиотеки *sklearn*.

```
# загружаем датасет load_iris
iris_dataset = load_iris()
```

3. Ознакомление с данными.

Просмотрите описание набора данных.

```
# получим информацию о датасете load_iris
print(iris_dataset.DESCR)
```

Датасет содержит информацию о 150 цветках ириса. Каждый цветок характеризуется 4 параметрами:

- длина чашелистика в сантиметрах (*sepal length*);
- ширина чашелистика в сантиметрах (*sepal width*);
- длина лепестка в сантиметрах (*petal length*);

– ширина лепестка в сантиметрах (*petal width*).

Все цветки ириса подразделяются на 3 класса:

– *setosa*;

– *versicolour*;

– *virginica*.

Выведите на экран параметры цветков ириса.

```
# выведем набор данных с параметрами цветков ириса
print(iris_dataset.data)
```

Каждый столбец набора данных характеризует тот или иной параметр цветка ириса: длина чашелистика, ширина чашелистика, длина лепестка, ширина лепестка.

Каждая строка набора данных – это информация о параметрах одного цветка ириса.

Отобразите метки классов.

```
# выведем метки цветков ириса
print(iris_dataset.target)
```

Метка показывает, к какому классу относится тот или иной цветок ириса. Все цветки распределены по трем классам. Метка «0» означает, что цветок ириса относится к классу «*setosa*», метка «1» – к классу «*versicolour*», метка «2» – к классу «*virginica*».

Обратите внимание, цветки ириса выстроены в датасете в строгом порядке: в начале списка идут цветки с меткой «0» (класс «*setosa*»), вторыми перечислены цветки с меткой «1» (класс «*versicolour*»), третьи в списке – цветки с меткой «2» (класс «*virginica*»).

4. Разделение данных на обучающую и тестовую выборки.

Процесс построения модели разбивается на два основных этапа:

– этап обучения модели;

– этап тестирования модели (проверка надежности модели).

Соответственно набор данных о цветках ириса нужно разбить на два подмножества. Большее по количеству объектов подмножество используется для обучения модели. На меньшем подмножестве осуществляется проверка качества обучения модели.

Разобьем датасет с данными о цветках ириса на обучающую и тестируемую выборку. Обозначим набор данных обучающей выборки следующим образом: данные с характеристиками цветков ириса – *X_train*, соответствующие им метки – *y_train*. Набор данных тестируемой выборки обозначим следующим образом: данные с характеристиками цветков ириса – *X_test*, соответствующие им метки – *y_test*.

Для разбиения датасета в библиотеке *scikit-learn* есть функция *train_test_split*(*X*, *y*, *train_size* = *k*, *random_state* = 10), где *X* – данные с характеристиками объектов, *y* – метки объектов, *k* – доля датасета, которую нужно включить в обучающую выборку.

Одновременно функция *train_test_split*(*X*, *y*, *train_size* = *k*, *random_state* = 10) перемешивает набор данных в датасете.

Разделите набор данных с использованием функции *train_test_split*, значение переменной *variant_number* должно быть равно номеру вашего варианта.

```
# разбиваем датасет на обучающую и тестируемую выборку
variant_number = 15 # номер варианта
k = 1 - ((variant_number + 10) / 100) # пропорция обучающей выборки
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset.data, iris_dataset.target, train_size=k, random_state=10
)
```

Проверьте размеры обучающей и тестовой выборок.

```
# Проверяем размер выборок
print(f"Размер обучающей выборки: {len(X_train)}, {len(y_train)}")
print(f"Размер тестовой выборки: {len(X_test)}, {len(y_test)}")
```

5. Выбор и обучение моделей классификации.

Обучим разрабатываемую модель для решения задачи классификации объектов с помощью трех алгоритмов классификации: *kNN* (метод *k* ближайших соседей), *Logistic Regression* (метод логистической регрессии), *Random Forest* (метод случайных деревьев).

Метод *kNN* (метод *k* ближайших соседей) был подробно описан в предыдущей лабораторной работе.

Метод логистической регрессии является методом классификации, в соответствии с которым определяется вероятность отнесения исходного значения к тому или иному классу.

Результатом использования логистической регрессии является построение *n*-мерной разделительной плоскости, разделяющей пространство исходных значений на классы.

Предположим, множество исходных чисел состоит из двух классов: положительных и отрицательных чисел.

Если логистическая функция принимает значение от 0,5 до 1, то исходное число относится к классу положительных чисел. Если функция принимает значение от 0 до 0,5, то исходное число относится к классу отрицательных чисел (рис. 1).

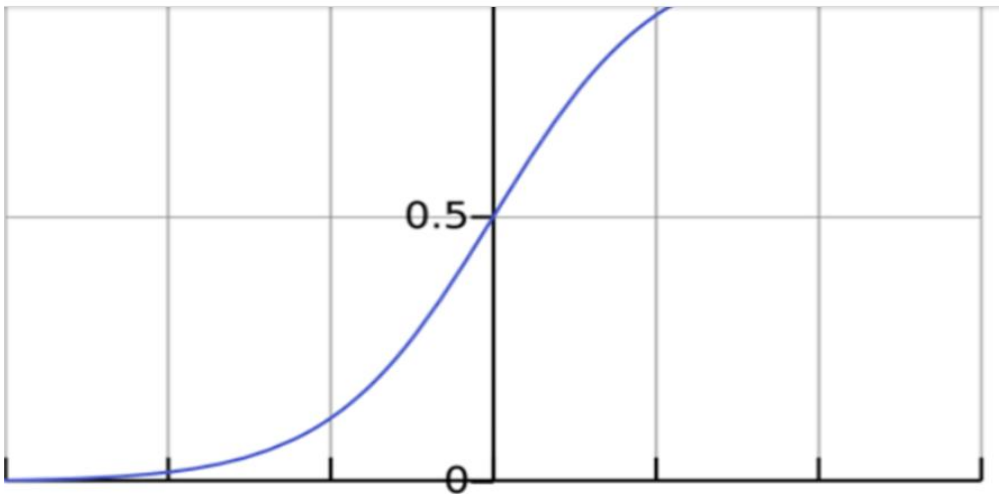

Рис. 1. График логистической регрессии

Загрузите алгоритмы классификации *kNN*, *Logistic Regression*, *Random Forest* в новую кодовую ячейку. Количество ближайших соседей *n* в алгоритме классификации *kNN* установите равным 5.

```
# инициализируем классификаторы
knn = KNeighborsClassifier(n_neighbors=5)
logreg = LogisticRegression(max_iter=1000)
ranfor = RandomForestClassifier()
```

Обучите модели на обучающей выборке.

Для обучения моделей воспользуйтесь функцией *fit* из библиотеки *sklearn*. Данная функция подает на входной слой нейросети данные о характеристиках объектов из обучающей выборки (параметры цветков ириса), а на выходной слой – метки данных объектов (классы цветков ириса). Обучение модели заключается в установлении таких параметров нейросети, которые обеспечивают соответствие между характеристиками объектов и их метками.

```
# обучаем модели
knn.fit(X_train, y_train)
logreg.fit(X_train, y_train)
ranfor.fit(X_train, y_train)
# делаем предсказания
y_knn = knn.predict(X_test)
y_logreg = logreg.predict(X_test)
y_ranfor = ranfor.predict(X_test)
# выводим результаты
print("Фактические метки:", y_test)
print("kNN:", y_knn)
print("Logistic Regression:", y_logreg)
print("Random Forest:", y_ranfor)
```

6. Оценка качества моделей.

Оцените качество моделей с помощью функции *score*. Данная функция сравнивает метки, предсказанные моделью для объектов тестовой выборки, с фактическим метками данных объектов и рассчитывает долю правильно предсказанных меток. Максимальное значение показателя *score* равно 1. Это означает, что все метки для объектов тестовой выборки предсказаны правильно.

Метрика *score* считает долю правильно предсказанных объектов, а для более точной оценки лучше использовать метрики *precision*, *recall*, *F*1-*score*.

```
# оцениваем качество моделей
print("kNN score:", knn.score(X_test, y_test))
print("Logistic Regression score:", logreg.score(X_test, y_test))
print("Random Forest score:", ranfor.score(X_test, y_test))
```

Результат оценки качества моделей представлен на рис. 2.

```
[4]  # Оценка качества моделей
     print("kNN score:", knn.score(X_test, y_test))
     print("Logistic Regression score:", logreg.score(X_test, y_test))
     print("Random Forest score:", ranfor.score(X_test, y_test))

     kNN score: 0.9736842105263158
     Logistic Regression score: 1.0
     Random Forest score: 0.9736842105263158
```

Рис. 2. Проверка качества обучения моделей классификации с помощью метрики *score*

7. Классификация новых объектов.

Создайте данные для новых объектов и выполните их классификацию.

Так как каждый цветок ириса характеризуется 4 параметрами, то для каждого цветка укажите 4 числа.

Для первого цветка первое число должно совпадать с порядковым номером дня из даты рождения студента. Для второго цветка первое число должно совпадать с порядковым номером месяца из даты рождения студента.

```
# классификация новых объектов
```

```
day = 31  # заменить на порядковый номер своего дня рождения
month = 12  # заменить на порядковый номер своего месяца рождения
X_new = [[day, 4, 3, 1], [month, 9, 3, 10]]
# Предсказания для новых объектов
print("kNN:", iris_dataset['target_names'][knn.predict(X_new)])
print("Logistic Regression:", iris_dataset['target_names'][logreg.predict(X_new)])
print("Random Forest:", iris_dataset['target_names'][ranfor.predict(X_new)])
```

Результат классификации представлен на рис. 3.

```
[5]  # Корректные данные для классификации новых объектов
     X_new = [[31, 4, 3, 1], [12, 9, 3, 10]]  # Примеры из диапазона признаков ирисов

     # Предсказания для новых объектов
     print("kNN:", iris_dataset['target_names'][knn.predict(X_new)])
     print("Logistic Regression:", iris_dataset['target_names'][logreg.predict(X_new)])
     print("Random Forest:", iris_dataset['target_names'][ranfor.predict(X_new)])

     kNN: ['virginica' 'virginica']
     Logistic Regression: ['versicolor' 'virginica']
     Random Forest: ['versicolor' 'versicolor']
```

Рис. 3. Классификация новых цветков ириса

8. Сохранение работы.

Сохраните блокнот через меню «Файл / Сохранить».

Переименуйте файл на *Google* Диске, используя шаблон «Лабораторная работа № 3 студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

Задание

В задании используются следующие условные обозначения:

– N – порядковый номер первой буквы фамилии студента согласно русскому алфавиту;

– M – порядковый номер первой буквы имени студента согласно русскому алфавиту.

Разработайте модели классификации для набора данных *load_wine*.

При этом должны выполняться следующие параметры:

– доля объектов, включаемых в тестируемую выборку, должна равняться $N + 5$;

– количество ближайших соседей в методе *kNN* должно равняться 7;

– в наборе данных для двух новых объектов для первой бутылки вина первое число должно совпадать с *N*;

– в наборе данных для двух новых объектов для второй бутылки вина первое число должно совпадать с *M*.

Контрольные вопросы

1. Что такое машинное обучение, и какие основные этапы разработки модели вы можете назвать?

2. В чём состоит основная цель задачи классификации?

3. Какие этапы включает в себя процесс построения модели машинного обучения?

4. Что такое обучающая и тестовая выборки, и зачем их разделяют?

5. Как загрузить встроенные датасеты из библиотеки *scikit-learn*?

6. В чём заключается принцип работы метода ближайших соседей (*kNN*)? Какие параметры важно учитывать при его использовании?

7. Объясните, как работает метод логистической регрессии. Почему он подходит для решения задач классификации?

8. Какой принцип лежит в основе метода случайного леса (*Random Forest*)? Какие преимущества он имеет?

9. Что означает значение *score*, возвращаемое моделью? Как интерпретировать его?

10. Почему важно тестировать модель на данных, которые она раньше не видела?

Содержание отчета

1. Титульный лист

2. Цель работы.

3. Формулировка задания.

4. Описание результатов выполнения задания в текстовом и графическом виде (скриншоты).

5. Ответы на контрольные вопросы.

6. Подробный вывод о проделанной работе.

Лабораторная работа № 4
РАЗРАБОТКА МОДЕЛИ ЛИНЕЙНОЙ РЕГРЕССИИ

Время выполнения – 8 часов (аудиторная работа – 2 часа, самостоятельная работа – 6 часов).

Цель работы: получить навыки прогнозирования количественных признаков объекта (например, цены недвижимости, стоимости подержанных автомобилей, экономических показателей, прогноза состояния пациента и т. д.) на основе построения и оценки качества моделей регрессии.

Задачи работы

1. Изучить библиотеки, модули и фреймворки *Python*, применяемые для решения задач регрессии.

2. Освоить основные алгоритмы регрессии.

3. Научиться разрабатывать, обучать и оценивать модели регрессии на языке *Python* в *Google Colab*.

Перечень обеспечивающих средств

1. Платформа *Google Colab*.

2. Библиотеки, модули и фреймворки *Python*, предназначенные для решения задач регрессии.

Общие теоретические сведения

Построение модели машинного обучения зависит от задачи, которую данная модель должна решать. Выделяют три основные задачи:

– классификация (рассмотрена в предыдущих лабораторных работах);

– регрессия;

– кластеризация.

Регрессия (*regression*) – это задача предсказания для каждого объекта значения, которое является действительным числом (например, прогноз стоимости акций, объема продаж товара, количества загрузок мобильного приложения и т. д.).

Предположим, у объектов (красные точки) имеется единственный признак *x*. Тогда ось абсцисс будет соответствовать этому признаку *x*, а ось ординат – целевому значению *y*, зависящему от переменной *x*.

Прямая линия – это построенная модель регрессии, цель которой – проходить максимально близко ко всем точкам. Построив модель регрессии, можно спрогнозировать целевое значение *y* при определенном значении переменной *x* (рис. 1).

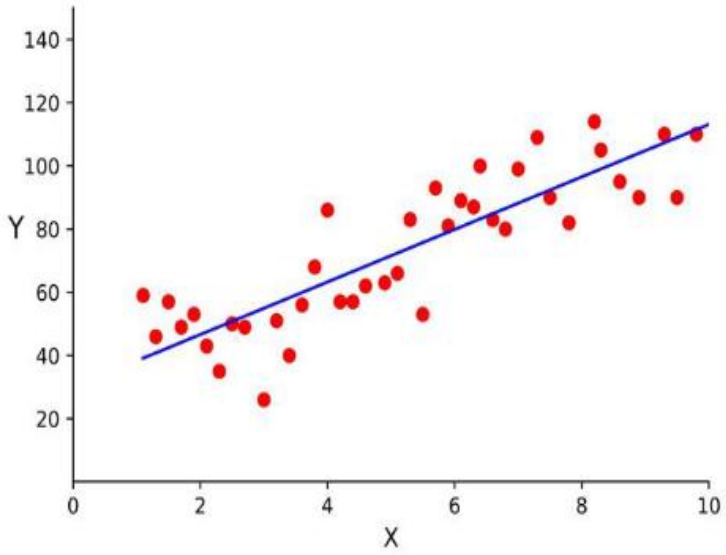

Рис. 1. Линейная регрессия

1. Создание нового блокнота.

Создайте новую текстовую ячейку и впишите следующий текст: «Лабораторная работа студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

Установите пакет *scikit-learn*. Если библиотека уже установлена, этот шаг можно пропустить.

```
# загружаем библиотеку scikit-learn
!pip install scikit-learn
```

2. Модель для прогнозирования цен на недвижимость на основе датасета *fetch_california_housing*.

Для создания модели потребуются модули, фреймворки и датасеты библиотеки *scikit-learn*.

Загрузите необходимые для работы библиотеки.

```
# Загружаем библиотеку scikit-learn
import numpy as np
import sklearn
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
```

3. Загрузка датасета *fetch_california_housing*.

Загрузите данный датасет из модуля *datasets* библиотеки *sklearn* и обозначьте его как *housing*.

```
# загружаем датасет
housing = fetch_california_housing()
X, y = housing.data, housing.target
```

4. Изучение информации о датасете.

Ознакомьтесь с информацией о данном датасете, воспользовавшись таким параметром датасета как ‘*DESCR’*.

```
# информация о датасете housing
print(housing.DESCR)
```

Датасет содержит информацию о жилой недвижимости в Калифорнии. Каждый объект характеризуется 8 параметрами:

– *MedInc* – медианный доход жителей района (в десятках тысяч долларов);

– *HouseAge* – средний возраст домов в районе;

– *AveRooms* – среднее количество комнат на домохозяйство;

– *AveBedrms* – среднее количество спален на домохозяйство;

– *Population* – численность населения района;

– *AveOccup* – среднее количество жителей на домохозяйство;

– *Latitude* – географическая широта района;

– *Longitude* – географическая долгота района.

Целевая переменная (*MedHouseVal*) – медианная стоимость дома в районе (в сотнях тысяч долларов).

5. Вывод набора данных с параметрами объектов недвижимости.

Выведите на экран набор данных с параметрами объектов недвижимости.

```
# выводим набор данных с параметрами 10 объектов недвижимости
print(housing.data[0:10])
```

Каждый столбец в наборе данных характеризует параметры объекта недвижимости (например, средний возраст домов, численность населения района и др.). Каждая строка представляет параметры одного объекта недвижимости.

6. Вывод цен 10 объектов недвижимости.

Выведите на экран цены 10 объектов недвижимости (метки).

```
# выводим метки объектов недвижимости
print(housing.target[0:10])
```

7. Построение модели регрессии для предсказания цен объектов недвижимости.

Процесс делится на два этапа:

- обучение модели;
- тестирование модели (оценка надежности).

Набор данных нужно разделить на обучающую и тестовую выборки:

- обучающая выборка содержит большую часть данных и используется для обучения;
- тестовая выборка служит для проверки качества модели.

Разбиение выполняется функцией *train_test_split*(*X*, *y*, *train_size* = *k*, *random_state* = 10), где *X* – данные с характеристиками объектов, *y* – метки объектов, *k* – доля датасета. Функция перемешивает данные и разделяет их в заданных пропорциях.

Разбейте датасет с данными об объектах недвижимости на обучающую и тестовую выборки. Обозначьте набор данных обучающей выборки следующим образом: данные с характеристиками объектов недвижимости – *X_train*, соответствующие им метки – *y_train*. Набор данных тестовой выборки обозначьте следующим образом: данные с характеристиками объектов недвижимости – *X_test*, соответствующие им метки – *y_test*.

Проверьте правильность разбиения датасета с помощью функции len, которая подсчитывает количество данных в массиве.

```
# разбиваем датасет на обучающую и тестовую выборки
variant_number = 15  # замените на свой номер варианта
k = 1 - ((variant_number + 10) / 100)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=k, random_state=10)
# проверяем размеры выборок
print(f"Размер обучающей выборки: {len(X_train)} объектов")
print(f"Размер тестовой выборки: {len(X_test)} объектов")
```

8. Выбор алгоритма для обучения модели.

Рассмотрим два алгоритма регрессии:

– линейная регрессия;

– полиномиальная регрессия.

9. Линейная регрессия.

Уравнение линейной регрессии:

$$a(x_i) = w_0 + w_1 * x_{i1} + \cdots + w_d * x_{id}, \quad (1)$$

где $a(x_i)$ – регрессионная модель,

$x_i = (x_{i1}, \dots, x_{id})$ – входной вектор, содержащий d значений признаков x_{ij},

$w_0, w_1, \dots, w_d$ – весовые коэффициенты (веса), полностью определяющие предсказания модели.

Обучение модели заключается в нахождении таких коэффициентов, при которых минимизируется ошибка. После обучения модель способна предсказывать значение целевой переменной для любого входного вектора.

10. Полиномиальная регрессия.

Для построения полиномиальной регрессии нужно ввести новые признаки, построив их на основе существующих. В полиномиальной регрессии вводятся новые признаки с помощью полинома: $w_0 + w_1 * x + w_2 * x^2 + \dots + w_n * x^n$.

Степень полинома *n* определяет степень нелинейности модели (рис. 2). Увеличение степени полинома приводит к более гибкому моделированию, но может вызвать переобучение.

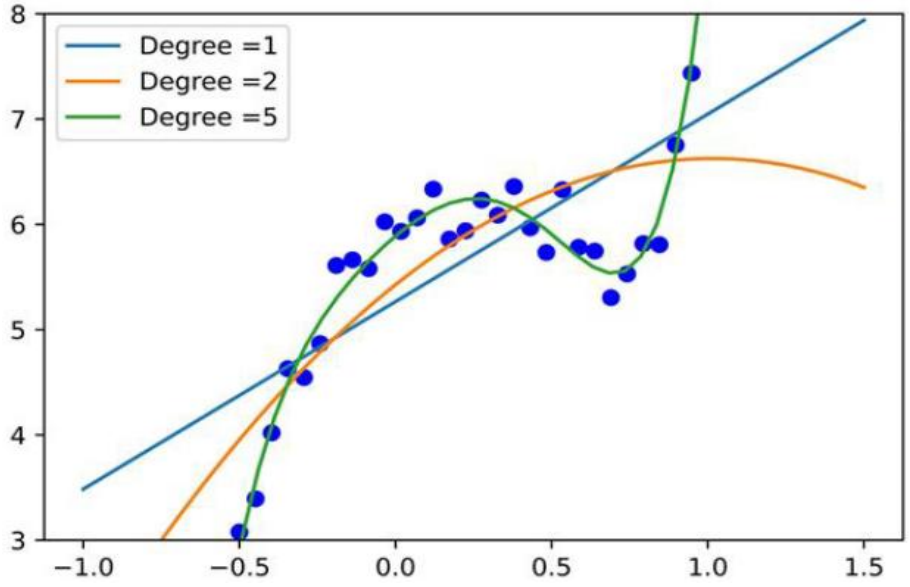

Рис. 2. Полиномиальная регрессия

11. Обучение и тестирование моделей.

Обучите и протестируйте две модели (линейная регрессия, полиномиальная регрессия). Используйте функции *fit* (обучение) и *predict* (тестирование).

Для обучения моделей воспользуйтесь функцией *fit* из библиотеки *sklearn*. Функция *fit* подает на вход обучающей выборки данные о характеристиках объектов и соответствующие им метки.

Для тестирования моделей воспользуйтесь функцией *predict* из библиотеки *sklearn*. Данная функция подает на входной слой нейросети данные о характеристиках объектов из тестируемой выборки (параметры объектов недвижимости), но, в отличие от функции *fit*, не подает на выходной слой метки данных объектов (цены продажи этих объектов). Модель сама должна спрогнозировать цены объектов недвижимости из тестируемой выборки.

```
# линейная регрессия
# создаем и обучаем модель
LR = LinearRegression()
LR.fit(X_train, y_train)
# делаем предсказание
y_pred_LR = LR.predict(X_test)
# полиномиальная регрессия
# преобразуем признаки
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
# обучаем модель
LR_poly = LinearRegression()
LR_poly.fit(X_train_poly, y_train)
# делаем предсказание
y_pred_poly = LR_poly.predict(X_test_poly)
```

12. Сравнение моделей.

Выведите на экран фактические цены, по которым были проданы *k* первых объектов недвижимости из тестируемой выборки, и цены продажи этих объектов, предсказанные разработанными моделями. Установите значение *k* равным номеру варианта.

```
# выведем предсказанные и фактические цены
k = variant_number  # количество объектов для сравнения (номер варианта)
print("Фактические цены:", y_test[:k])
print("Линейная регрессия:", y_pred_LR[:k])
print("Полиномиальная регрессия:", y_pred_poly[:k])
```

13. Оценка качества моделей.

Воспользуйтесь функцией *mean_squared_error* (среднеквадратичная ошибка, *MSE*). Функция рассчитывает среднее арифметическое квадратов разностей между предсказанными моделью регрессии ценами на объекты недвижимости из тестируемой выборки фактическими ценами продажи этих объектов, содержащимися в датасете. Чем ниже значение показателя, тем надежнее модель, тем лучше она предсказывает цены недвижимости. Если *MSE* полиномиальной регрессии значительно ниже, чем у линейной, это может указывать на лучшее приближение данных, но при слишком малой тестовой выборке — на переобучение.

```
# оценим качество моделей
mse_LR = mean_squared_error(y_test, y_pred_LR)
mse_poly = mean_squared_error(y_test, y_pred_poly)
print("MSE линейной регрессии:", mse_LR)
print("MSE полиномиальной регрессии:", mse_poly)
```

14. Предсказание для новых объектов.

Создайте данные для двух новых объектов и выполните предсказание.

Так как каждый объект недвижимости характеризуется 8 параметрами, то для каждого объекта недвижимости укажите 8 чисел.

Для первого объекта недвижимости второй показатель, отражающий средний возраст домов в районе, должен совпадать с номером варианта. Для второго объекта этот же показатель установите равным величине 10 + номер варианта.

```
# предсказание для новых объектов недвижимости
X_new = np.array([
    [3.5, variant_number, 6, 1.2, 1200, 3.5, 37.4, -122.1],
    [4.0, 10 + variant_number, 7, 1.5, 1500, 4.0, 38.1, -121.8]
])
print("Предсказанные цены (Линейная регрессия):", LR.predict(X_new))
print("Предсказанные цены (Полиномиальная регрессия):", LR_poly.predict(poly.transform(X_new)))
```

15. Сохранение работы.

Сохраните блокнот через меню «Файл / Сохранить».

Переименуйте файл на *Google* Диске, используя шаблон «Лабораторная работа № 4 студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

Задания

В заданиях используются следующие условные обозначения:

– N – порядковый номер первой буквы фамилии студента согласно русскому алфавиту;

– M – порядковый номер первой буквы имени студента согласно русскому алфавиту.

1. В моделях регрессии для набора данных, состоящего из объектов недвижимости (датасет *fetch_california_housing*), выполните следующие условия:

– доля объектов, включаемых в тестируемую выборку, должна равняться N;

– количество объектов тестируемой выборки, отображаемых при выводе фактических цен продажи и предсказанных моделями цен, должно быть равно N;

– в наборе данных для двух новых объектов недвижимости:

1) для первого объекта второй показатель должен совпадать с N;
2) для второго объекта второй показатель установите равным величине 10 + M.

2. Разработайте на основе регрессии факторную модель прогнозирования количественной оценки состояния больного сахарным диабетом через год после определения исходного уровня заболевания (датасет *load_diabetes*).

При этом должны выполняться следующие условия:

– доля объектов, включаемых в тестируемую выборку, должна равняться N.

– в наборе данных для нового объекта (больного сахарным диабетом) значение первого параметра (возраст больного) должно совпадать с N.

Контрольные вопросы

1. Что такое регрессия? В каких задачах она применяется?
2. Чем задачи регрессии отличаются от классификации и кластеризации?
3. Как выглядит уравнение линейной регрессии?
4. В чем преимущества и недостатки полиномиальной регрессии?
5. Каково назначение функции mean_squared_error?
6. Какова структура датасета *fetch_california_housing*? Какие параметры и целевая переменная в нем представлены?
7. Как с помощью функции *train_test_split* задать размер тестируемой выборки?
8. Как выполняется обучение модели линейной регрессии с помощью функции *fit*?
9. В чем разница в использовании функции *predict* для моделей линейной и полиномиальной регрессии?
10. Как оценить качество модели регрессии? Какие метрики, кроме *MSE*, можно использовать?

Содержание отчета

1. Титульный лист
2. Цель работы.
3. Формулировка задания.

4. Описание результатов выполнения задания в текстовом и графическом виде (скриншоты).

5. Ответы на контрольные вопросы.

6. Подробный вывод о проделанной работе.

Лабораторная работа № 5
РАЗРАБОТКА МОДЕЛИ ДЛЯ КЛАСТЕРИЗАЦИИ МНОЖЕСТВА НЕПОМЕЧЕННЫХ ОБЪЕКТОВ

Время выполнения – 8 часов (аудиторная работа – 2 часа, самостоятельная работа – 6 часов).

Цель работы: получить навыки подготовки данных для машинного обучения и решения задач «без учителя» на примере кластеризации (разделения на кластеры) непомеченных объектов (объектов без меток).

Задачи работы

1. Освоить работу с библиотеками, модулями и фреймворками *Python*, используемыми для решения задач кластеризации непомеченных объектов.
2. Получить навыки конвертации файлов формата *.xlsx* в датафреймы, используемые в машинном обучении.
3. Научиться проводить подготовку и очистку данных в датафрейме перед обучением модели (приведение данных к единому формату, выявление и исправление недостоверных данных).
4. Изучить основные алгоритмы кластеризации (метод *k*-средних, графовый алгоритм Крускала, иерархическая кластеризация и др.)
5. Осуществить кластеризацию множества непомеченных объектов с использованием изученных алгоритмов.

Перечень обеспечивающих средств

1. Платформа *Google Colab*.
2. Платформа с наборами данных для машинного обучения *Kaggle*.
3. Библиотеки, модули и фреймворки *Python*, предназначенные для решения задачи кластеризации.

Общие теоретические сведения

Сущность машинного обучения «без учителя» заключается в том, что алгоритму до начала обучения не демонстрируются примеры и не предъявляются правильные ответы, так как их попросту нет. Например, имеется множество изображений объектов, которые потенциально могут быть разделены на несколько классов. Однако, в отличие от задачи классификации, объекты не помечены, то есть заранее неизвестно, к какому классу принадлежит то или иное изображение. Соответственно, мы не можем обучить модель на правильных примерах и показать ей, к какому классу относится каждое изображение.

На практике с непомеченными объектами приходится работать гораздо чаще, чем с помеченными, что подчеркивает актуальность машинного обучения «без учителя».

Одной из самых популярных задач машинного обучения «без учителя» является кластеризация.

Кластеризация – это процесс разделения множества непомеченных объектов на подмножества. Каждое подмножество называется кластером. Внутри одного кластера объекты должны быть относительно похожи друг на друга, но при этом отличаться от объектов в других кластерах.

Основой кластеризации является гипотеза компактности или непрерывности: близкие объекты схожи и принадлежат одному кластеру. Близость объектов определяется расстоянием между ними: схожесть или различие объектов зависят от того, насколько они удалены друг от друга. Таким образом, задача кластеризации заключается в том, чтобы распределить объекты по кластерам так, чтобы каждый кластер содержал близко расположенные объекты, а объекты из разных кластеров находились далеко друг от друга.

После разделения объектов на кластеры каждому объекту присваивается метка того кластера, к которому он принадлежит. Новый объект будет относиться к тому кластеру, в котором находятся ближайшие к нему объекты выборки.

Для выполнения кластеризации множество непомеченных объектов преобразуется в матрицу «объекты – признаки» размерностью *k* строк на *n* столбцов, где *k* – количество объектов в выборке, а *n* – число признаков, описывающих каждый объект.

1. Конвертация файлов *xlsx* в датафрейм.

Перейдите в раздел *Datasets* / *Classification* и найдите набор данных «*Student Stress Factors*: *A Comprehensive Analysis*» или воспользуйтесь прямой ссылкой: https://www.kaggle.com/datasets/rxnach/student-stress-factors-a-comprehensive-analysis.

Скачайте архив набора данных на локальный диск и разархивируйте его. В случае невозможности скачать набор данных с сайта *Kaggle.com* вы можете загрузить его по ссылке: https://drive.google.com/file/d/1MP1Y8vhwqSBM7e_3HPqiLVZp93jgDDfe/view?usp=sharing.

Этот файл содержит данные о 20 факторах, влияющих на уровень стресса студентов, и включает информацию о 1100 студентах.

Примечание: *CSV*-файл – текстовый файл, где строки таблицы разделены символами-разделителями (например, запятой).

Конвертируйте файл *CSV* в формат *xlsx*. Для этого выполните следующие действия:

– откройте файл *StressLevelDataset.csv* в *Excel*;

– выделите первый столбец;

– перейдите во вкладку «Данные» → «Текст по столбцам»;

– выберите формат данных «С разделителями»;

– укажите запятую в качестве разделителя;

– сохраните файл в формате *xlsx*, указав имя файла по шаблону «*Familiya_stress*», где *Familiya* – фамилия студента на английском языке.

Требования к данным в *Excel*:

– первая строка таблицы зарезервирована для заголовков, а первый столбец используется для идентификации единицы выборки;

– пробелы в названиях и значениях заменяются символами _, -, или написанием слов с заглавной буквы;

– имена столбцов и значений должны быть короткими и не содержать специальных символов (? $ % ^ & * ( ) - # < > / | \ [ ] { });

– все комментарии должны быть удалены.

2. Создание нового блокнота.

В новом блокноте создайте текстовую ячейку и впишите следующий текст: «Лабораторная работа студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

3. Загрузка библиотек и данных.

Для создания и работы с датафреймами используются модули и фреймворки *Python*, такие как библиотека *Pandas*.

Pandas – это библиотека *Python* для обработки и анализа структурированных данных. Название библиотеки происходит от выражения «*panel data*» («панельные данные»), что означает данные, организованные в виде таблиц. *Pandas* поддерживает операции чтения и записи данных в различных форматах, включая *Excel* 2007+, *CSV*, *SQL*, *HTML*, *JSON* и другие.

В библиотеке *Pandas* используются два основных класса объектов для работы с данными:

- *Series* – одномерный массив для хранения значений любого типа данных. Пример Series: [1, 2, 3, 1, 2, 3];
- *DataFrame* – двумерный массив, представленный в виде таблицы, где столбцы являются объектами класса Series.

Пример *DataFrame*:

	A	*B*	*C*	*D*
0	10	20	30	40
1	23	43	55	65
2	10	20	30	40

Загрузите библиотеку *Pandas*.

```
# загружаем библиотеку Pandas
```

```
import pandas as pd
```

Загрузите файл *smirnov_stress.xlsx* в *Google Colab*. Для этого сохраните данный файл на *Google* Диске и подключите *Google Диск* к среде *Google Colab*.

```
# подключаем хранилище Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

Укажите путь к файлу на *Google* Диске.

```
# укажем путь к файлу
smirnov_stress = '/content/drive/MyDrive/smirnov_stress.xlsx'
```

Конвертируйте файл в *DataFrame*.

Для выполнения данной операции используйте функцию библиотеки *Pandas pd.ExcelFile*(*file*). В скобках вместо *file* укажите название загруженного в среду *Google Colab* файла с информацией об уровне стресса студентов.

```
# конвертируем файл Excel в DataFrame
smirnov_stress_xl = pd.ExcelFile(smirnov_stress)
```

Проверьте наличие листов в файле.

```
# выводим на экран названий листов
smirnov_stress_xl.sheet_names
```

Загрузите данные из листа в *DataFrame*.

```
# загружаем данные из листа в DataFrame
smirnov_fr = smirnov_stress_xl.parse('StressLevelDataset')
```

Выведите первые *n* строк *DataFrame*, где n соответствует номеру варианта.

```
# выводим на экран первые n строк DataFrame
n = 25 # номер варианта
smirnov_fr.head(n)
```

4. Предварительная обработка данных.

В проектах по машинному обучению часто приходится работать с данными, собранными из различных источников. Как правило, такие данные содержат недостатки или пропуски, поэтому перед обучением модели требуется их предварительная обработка.

Подготовка данных – это важный этап в процессе машинного обучения, поскольку некорректные или «шумные» данные могут существенно снизить качество модели кластеризации.

Основная цель предварительной обработки данных – обеспечить их достаточное качество и корректность, чтобы построить надежную и эффективную модель кластеризации.

4.1. Проверьте формат данных.

При конвертации данных в *DataFrame* тип данных может быть распознан неверно, что потребует его изменения. Например, число «1» может быть загружено в датафрейм как строка (текстовый тип данных), а не как числовой.

Чтобы проверить типы данных в датафрейме, воспользуйтесь методом *df.info*().

```
# проверим формат данных датафрейма
smirnov_fr.info()
```

4.2. Исправьте неверный формат в столбце *blood_pressure*.

```
# исправим строчный формат данных на числовой формат
smirnov_fr['blood_pressure'] = pd.to_numeric(smirnov_fr['blood_pressure'], errors='coerce')
```

4.3. Исправьте отрицательные значения.

Периодически в датафреймах могут встречаться отрицательные значения в показателях, которые по определению должны быть только положительными, например, цена товара, объем продаж или возраст пациента.

В редактируемом датафрейме к таким показателям можно отнести, в частности, уровень кровяного давления студентов (столбец «*blood_pressure*»).

Найдите ошибочные данные с отрицательными значениями в столбце «*blood_pressure*» датафрейма, используя функцию *query*('*blood_pressure* < 0').

В случае, если были обнаружены отрицательные значения уровня кровяного давления студентов, исправьте их на положительные значения. Воспользуйтесь функцией *abs*(), отражающей абсолютное значение числа.

```
# найдем отрицательные значения уровень кровяного давления студентов
smirnov_fr.loc[smirnov_fr['blood_pressure'] < 0, 'blood_pressure'] = smirnov_fr['blood_pressure'].abs()
```

4.4. Удалите выбросы с помощью *Z*-оценки.

Выбросы – это результаты измерений, которые значительно выделяются из общей выборки данных. Они могут возникать, например, из-за погрешностей измерений датчиков или ошибок при ручном вводе данных.

Очистка датафрейма от случайных выбросов выполняется в два этапа:

– проверка объекта на наличие выбросов;

– обработка выявленных выбросов.

Методы обработки выбросов:

– замена на среднее значение;

– замена на медианное значение;

– замена на соседнее значение;

– трехточечная фильтрация, при которой каждое значение в ряду заменяется средним арифметическим данного значения и двух соседних.

Для выявления выбросов используется метод *Z*-оценки.

Z-оценка показывает, насколько текущее значение отклоняется от среднего в единицах стандартного отклонения.

Формула расчета *Z*-оценки:

$$z = \frac{X - \mu}{\sigma}$$

где X – текущее значение показателя,

μ – среднее значение,

σ – стандартное отклонение.

Значение считается выбросом, если его *Z*-оценка меньше −3 или больше +3.

Функция для расчета *Z*-оценки находится в библиотеке *SciPy*.

SciPy – это библиотека для *Python*, которая расширяет возможности библиотеки *NumPy*.

NumPy предназначена для работы с большими многомерными массивами данных и содержит базовые математические функции для операций с ними.

SciPy добавляет больше функций и методов для глубокого анализа данных.

Так как библиотека *SciPy* зависит от *NumPy*, для выполнения *Z*-оценки необходимо установить обе библиотеки.

```
# рассчитаем Z-оценку для определения случайных выбросов
import numpy as np
from scipy import stats
z_scores = np.abs(stats.zscore(smirnov_fr))
```

Если *Z*-оценка показателя больше или равна 3, то соответствующее значение считается выбросом и подлежит исключению из датафрейма.

Удалим строки, у которых хотя бы один признак имеет Z-оценку >= 3. В очищенном датафрейме должны остаться только значения, которые не являются выбросами.

```
# очистим датафрейм от случайных выбросов
smirnov_fr_clean = smirnov_fr[(z_scores < 3).all(axis=1)]
```

4.5. Проверьте наличие пропущенных данных.

```
# проверим наличие незаполненных ячеек
missing_data = smirnov_fr_clean.isnull().sum()
print(missing_data)
```

Если в датасете обнаружены незаполненные строки, следует либо исключить их (в случае небольшого количества пропусков), либо обработать одним из следующих способов:

– заменить пропущенные значения средним арифметическим по столбцу;

– заменить пропущенные значения медианой по столбцу;

– заменить пропущенные значения ближайшим соседним значением;

– применить трёхточечную фильтрацию: каждое значение в ряду числовых данных заменяется средним арифметическим этого значения и двух соседних.

4.6. Исключите столбец с метками объектов для кластеризации.

Полученный после этапа подготовки данных датафрейм готов для выполнения задач машинного обучения. Однако он содержит метки объектов, такие как уровень стресса (*stress_level*) для каждого студента. Поскольку кластеризация предполагает разделение множества объектов без заранее известных меток, необходимо подготовить данные.

Для этого удалите из датафрейма столбец, содержащий значения уровня стресса (*stress_level*).

Кроме того, удалите первую строку, которая содержит названия столбцов.

Выведите на экран первые 10 строк скорректированного датафрейма.

```
# удаляем столбец 'stress_level' из исходного датафрейма
smirnov_z1 = smirnov_fr_clean.drop(columns='stress_level')
# преобразуем датафрейм в массив NumPy (убираем индексы и заголовки столбцов)
smirnov_z2 = smirnov_z1.values
# выводим первые 10 строк массива для проверки
print(smirnov_z2[:10])
```

Теперь датафрейм полностью подготовлен для выполнения задачи кластеризации объектов.

5. Построение модели кластеризации.

Выберите алгоритм, с помощью которого будет обучаться разрабатываемая модель для решения задачи кластеризации объектов. Рассмотрите два алгоритма кластеризации:

– агломеративная иерархическая кластеризация;

– метод *k*-средних.

Рассмотрим алгоритм агломеративной иерархической кластеризации (алгоритм Ланса-Уильямса).

Изначально каждый объект рассматривается как отдельный кластер. На каждой итерации определяются два ближайших кластера, для которых межкластерное расстояние минимально среди всех возможных пар. Эти кластеры объединяются в один, формируя новое разбиение данных на кластеры. После этого межкластерные расстояния пересчитываются с учетом нового кластера, и алгоритм переходит к следующей итерации. Процесс продолжается до тех пор, пока все объекты не будут объединены в один общий кластер.

Для визуализации иерархической агломеративной кластеризации используется дендрограмма. На одной оси дендрограммы отображаются условные номера объектов выборки, а на другой – межкластерные расстояния. Линии от объектов ведут вертикально вверх вдоль оси межкластерных расстояний. Когда два кластера объединяются, между их линиями строится горизонтальная перемычка. Затем вместо двух линий от объединенных кластеров продолжается одна общая линия (рис. 5.1).

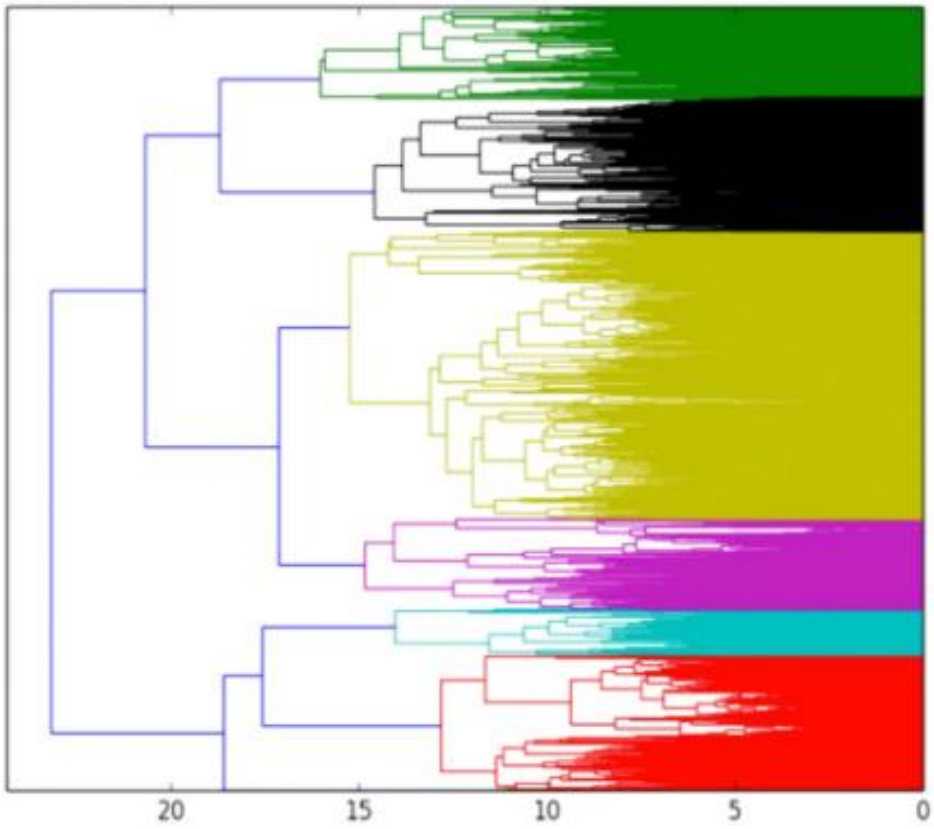

Рис. 5.1. Дендрограмма

5.1. Загрузите функцию *AgglomerativeClustering*. В качестве параметра данной функции укажите разделение множества объектов на 3 кластера.

```
# кластеризация с помощью алгоритма агломеративной иерархической кластеризации
import sklearn
from sklearn.cluster import AgglomerativeClustering
model = AgglomerativeClustering(n_clusters=3)
clusters = model.fit_predict(smirnov_z2)
```

5.2. Обучите модель на множестве *smirnov_z2* с использованием метода *fit_predict*. Также настройте вывод всех элементов массива с помощью *np.set_printoptions* и отобразите результаты кластеризации.

```
np.set_printoptions(threshold=np.inf)
smirnov_AgglClust = model.fit_predict(smirnov_z2)
print(smirnov_AgglClust)
```

5.3. Визуализируйте результаты кластеризации с помощью библиотеки *mglearn*.

```
# установим библиотеку mglearn
!pip install mglearn
```

Импортируйте модули библиотеки *mglearn* и функцию *scatter_matrix* для построения матрицы рассеяния с графиком *KDE* (графиком оценки плотности ядра вдоль диагоналей матрицы).

Датафрейм *smirnov_z1* содержит 20 столбцов (факторы, влияющие на уровень стресса студентов) и 1100 строк (данные о студентах). Визуализация результатов кластеризации такого большого набора данных обладает низкой информативностью. Поэтому создайте матрицу рассеивания только для первых четырех столбцов датафрейма и отобразите результаты кластеризации первых студентов в количестве, равном номеру вашего варианта + 200.

```
import mglearn
```

```
grafic = pd.plotting.scatter_matrix(
    smirnov_z1.iloc[0:219, 0:4],
    c=clusters[0:219],
    figsize=(15, 15),
    marker='o',
    diagonal='kde'
)
```

Теперь рассмотрим метод *k*-средних, который является одним из наиболее популярных методов кластеризации.

Пусть имеется выборка *X*, состоящая из непомеченных объектов, каждый из которых характеризуется n признаками. Заранее задается предполагаемое количество кластеров *k*.

Суть алгоритма заключается в следующем:

1. Выбираются начальные центры кластеров. Обычно это объекты, которые максимально удалены друг от друга.

2. Каждый объект из множества *X* назначается тому кластеру, центр которого находится к нему ближе всего (в зависимости от выбранной метрики расстояния, например, Евклидовой).

3. Для каждого кластера вычисляется суммарное квадратичное отклонение точек кластера от его текущего центра.

4. Проверяется, есть ли внутри кластера объекты, которые могут стать новым центром, минимизирующим суммарное квадратичное отклонение точек кластера. Если такой объект найден, он назначается новым центром кластера.

5. После обновления центров кластеров происходит перераспределение объектов множества *X* между кластерами.

Итерации продолжаются до тех пор, пока на очередной итерации не перестанет изменяться внутрикластерное расстояние (суммарное отклонение). Распределение объектов по кластерам на этой итерации считается окончательным (рис. 5.2).

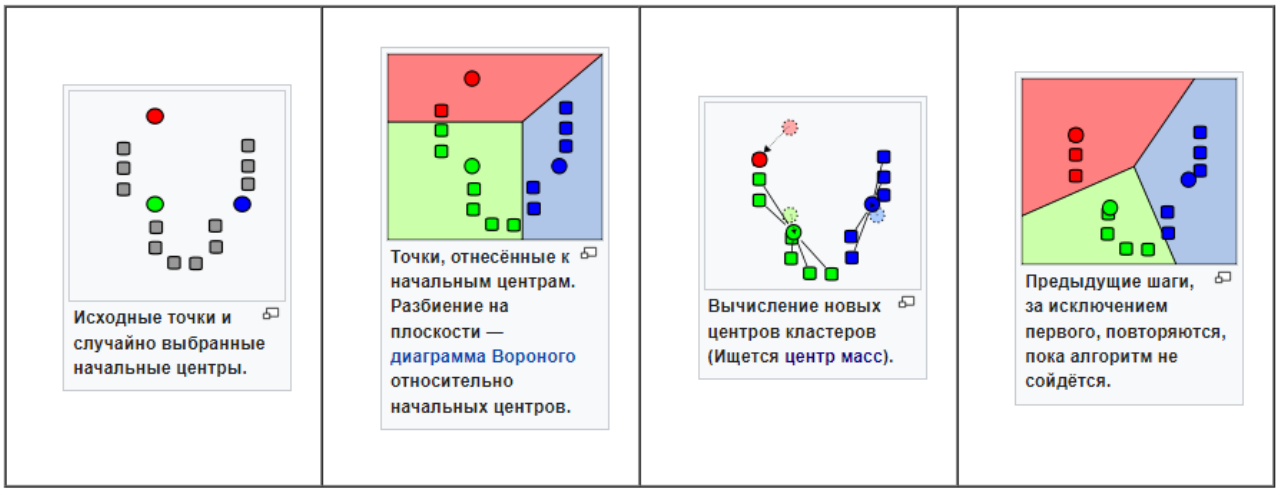

Рис. 5.2. Метод *k*-средних

5.4. Обучите модель кластеризации методом *k*-средних.

```
# метод k-средних
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0, n_init=10)
kmeans_clusters = kmeans.fit_predict(smirnov_z2)
```

5.5. Выполните кластеризацию множества с использованием метода *k*-средних.

```
# разделим множество на кластеры с помощью метода k-средних
y_predict_kmeans = kmeans.predict(smirnov_z2)
print(y_predict_kmeans)
```

6. Создание нового объекта.

Создайте новый объект с параметрами факторов, определяющих уровень стресса нового студента.

Факторы, определяющие уровень стресса студента:

– самооценка;

– уровень тревожности;

– история психического здоровья;

– депрессия;

– головная боль;

– кровяное давление;
– качество сна;
– проблемы с дыханием;
– уровень шума;
– условия жизни;
– безопасность;
– основные потребности;
– академическая успеваемость;
– учебная нагрузка;
– отношения между преподавателем и студентом;
– забота о будущей карьере;
– социальная поддержка;
– давление со стороны сверстников;
– внеклассные мероприятия;
– травля.

```
# создаем новый объект
variant_number = 15 # укажите ваш номер варианта
New_student = [variant_number, variant_number + 10, 0, 11, 2, 1, 2, 4, 2, 3, 3, 2, 3, 2, 3, 3, 2, 3, 3, 2]
```

Первый параметр (самооценка) установите равным номеру вашего варианта.

Второй параметр (уровень тревожности) сделайте равным номеру варианта + 10.

С помощью обученной модели *K-Means* выполните предсказание для нового объекта.

```
# определим метку нового объекта
New_student_predict_KMeans = kmeans.predict([New_student])
print(New_student_predict_KMeans)
```

7. Сохранение работы.

Сохраните блокнот через меню «Файл / Сохранить».

Переименуйте файл на *Google* Диске, используя шаблон «Лабораторная работа № 5 студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

Задания

В заданиях используются следующие условные обозначения:

– N – порядковый номер первой буквы фамилии студента согласно русскому алфавиту;

– M – порядковый номер первой буквы имени студента согласно русскому алфавиту.

1. В модели кластеризации уровня стресса студентов настройте параметры выполнения следующим образом:

– при выводе строк *DataFrame* с помощью команды *head*(n) количество выводимых строк (n) должно быть равно N;

– при построении матрицы рассеивания на экран должны выводиться результаты кластеризации первых студентов в количестве $N + 200$.

Для нового объекта в наборе данных (определение уровня стресса нового студента) задайте:

– первый параметр (самооценка) – N,

– второй параметр (уровень тревожности) – M.

2. Разработайте модель кластеризации источников воды с точки пригодности ее для питья.

Выполните следующие шаги:

1. Скачайте с сайта *Kaggle* файл *Water Quality and Potability* в формате *CSV*, содержащий информацию о качестве источников воды.

Этот набор данных содержит измерения качества воды и оценки ее пригодности для потребления человеком. Каждая строка в наборе представляет собой образец воды с определенными количественными характеристиками.

Характеристики качества воды:

– *pH* – показатель концентрации ионов водорода;

– жесткость (*hardness*);

– примеси (*solids*);

– хлорамины (*chloramines*);

– сульфат (*sulfate*);

– электропроводность (*conductivity*);

– органический углерод (*organic_carbon*);

– тригалометаны (*trihalomethanes*);

– мутность (*turbidity*).

Столбец «Пригодность для питья» (*Potability*) содержит булевый признак, указывающий, пригодна ли вода для потребления.

2. Конвертируйте скачанный *CSV*-файл в формат *XLSX*.

3. Создайте на его основе объект *DataFrame*.

4. Выполните очистку и предварительную подготовку данных:

– проверьте и исправьте формат данных;

– исправьте отрицательные значения на положительные, если они обнаружены;

– удалите случайные выбросы (аномалии).

5. Удалите столбец *Potability*, содержащий метки (*target*), которые указывают, к какой группе по пригодности для питья относится источник воды.

6. Разделите источники воды на два кластера с использованием метода k-средних.

7. Визуализируйте результаты кластеризации, отобразив матрицу рассеивания для первых источников воды в количестве $N + 200$.

8. Создайте набор данных для нового объекта, чтобы оценить возможность потребления ее человеком. Для этого задайте параметр *pH* равным $M + 18$.

9. Определите, к какому кластеру относится новый источник воды.

Контрольные вопросы

1. Что представляет собой машинное обучение «без учителя»?
2. В чем заключается основная цель кластеризации объектов?
3. Как определяется схожесть объектов при кластеризации?
4. Какие этапы включает процесс подготовки данных для кластеризации?
5. В чем разница между алгоритмами кластеризации методом *k*-средних и агломеративной иерархической кластеризации?
6. Какие этапы включены в предварительную обработку данных?
7. Как проверить наличие пропущенных данных в датафрейме?
8. Какие методы существуют для устранения выбросов в данных?
9. Почему необходимо исключать столбцы с метками объектов при кластеризации?
10. Как работает метод агломеративной иерархической кластеризации?
11. Как работает метод *k*-средних? Опишите процесс его выполнения.
12. Почему важно правильно задавать количество кластеров при использовании метода *k*-средних?
13. Почему для кластеризации важно проводить предварительную обработку данных?

Содержание отчета

1. Титульный лист
2. Цель работы.
3. Формулировка задания.
4. Описание результатов выполнения задания в текстовом и графическом виде (скриншоты).
5. Ответы на контрольные вопросы.
6. Подробный вывод о проделанной работе.

Лабораторная работа № 6
РАЗРАБОТКА ЧАТ-БОТА НА ОСНОВЕ МАШИННОГО ОБУЧЕНИЯ

Время выполнения – 8 часов (аудиторная работа – 2 часа, самостоятельная работа – 6 часов).

Цель работы: освоение навыков разработки чат-бота, способного отвечать на вопросы пользователей по заранее заданной тематике.

Задачи работы

1. Изучить инструменты *Python* для обработки текстовых данных и создания чат-ботов.

2. Освоить создание *JSON*-файлов с использованием онлайн-редактора данных, содержащих: перечень намерений пользователей, задающих вопросы; возможные вопросы, соответствующие этим намерениям; набор ответов на вопросы.

3. Начать разработку итогового проекта по созданию специализированного чат-бота.

Перечень обеспечивающих средств

1. Платформа *Google Colab*.

2. Онлайн-редактор данных формата *JSON*.

3. Библиотеки, модули и фреймворки *Python*, предназначенные для обработки текстовых данных и разработки чат-ботов на основе машинного обучения.

Общие теоретические сведения

В рамках метода проектного обучения результатом освоения дисциплины становится созданный студентом чат-бот, способный отвечать на вопросы пользователей по заданной тематике.

Чат-бот – это программа, использующая технологии искусственного интеллекта, которая имитирует общение с человеком и может мгновенно реагировать на пользовательские запросы.

Проект, разработанный студентом, может найти практическое применение в учебной деятельности, например, при защите курсовых работ, отчетов по практике, а также при организации студенческих конференций и научных семинаров.

Реализация итогового проекта начинается с шестой лабораторной работы, в которой студенты разрабатывают чат-ботов на основе алгоритмов классификации, и продолжается в следующих лабораторных работах.

1. Создание нового блокнота.

Создайте новый блокнот в *Google Colab*, добавьте текстовую ячейку и впишите следующий текст: «Лабораторная работа студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

2. Работа с файлами формата *JSON*.

JSON (*JavaScript Object Notation*) – текстовый формат, предназначенный для обмена данными между программами. Он похож на словари и списки *Python*, но записывается как обычный текст.

Объекты *JSON* представляют собой пары «ключ-значение» и заключаются в фигурные скобки {}.

Ключи могут быть только строками, значения – различными типами данных.

Если ключ имеет несколько значений, они записываются в квадратные скобки [] через запятую.

```
students = {
    "Студенты": ["Иванов", "Петров", "Сидоров"]
}
```

В указанном примере «Студенты» – это ключ, а фамилии студентов («Петров», «Иванов») – это значения, которые принимает ключ

Создайте *JSON*-файл с использованием *JSON Editor Online* (*https://jsoneditoronline.org*) или другого редактора.

В левой части редактора *JSON Editor Online* наберите необходимый текст и трансформируйте (*Transform*) его в файл *JSON*.

Пример содержимого файла *JSON*:

```
{
  "Студенты": {
    "Фамилия": {
      "Вопрос": [
        "Ваша фамилия?",
        "Как Вас зовут?",
        "Представьтесь, пожалуйста."
      ],
      "Ответ": ["Иванов", "Петров", "Сидоров"]
    },
    "Дисциплины": {
      "Вопрос": [
        "Какие дисциплины Вы изучаете?",
        "Что Вам преподают?",
        "Чему Вас учат?"
      ],
      "Ответ": ["Математика", "История", "Физика"]
    }
  }
}
```

Одна из фамилий должна совпадать с фамилией студента, выполняющего лабораторную работу.

Обратите внимание, что у разных объектов («Фамилия», «Дисциплина») одинаковые ключи («Вопрос», «Ответ»), хотя значения ключей различны.

Сохраните файл *JSON* на *Google* Диск.

3. Загрузка данных в *Google Colab*.

Подключите *Google* Диск.

```
# подключаем Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

Найдите файл *JSON*, скопируйте его путь и загрузите в *Colab*:

```
# выбор файла в Google Drive
file_path = '/content/drive/MyDrive/StudentDis.json'
with open(file_path, "r", encoding="utf-8") as file:
    data = file.read()
```

Данные в файле *json* записаны как обычный текст. Чтобы текст в файле *json* конвертировался в объекты, необходимо из библиотеки *json* загрузить функцию *loads*(), которая преобразует текст в объект (пара «ключ-значение»).

Импортируйте библиотеку *json* и конвертируйте содержимое файла:

```
# импорт библиотеки json
import json
students_data = json.loads(data)
print(students_data)
```

4. Работа с массивами данных.

Массив – это упорядоченный набор элементов, каждый из которых имеет свой уникальный номер (индекс), позволяющий быстро получить к нему доступ. Нумерация элементов в массиве начинается с 0.

Для загрузки объектов из файла *JSON* в массив можно воспользоваться методом *append*(). Этот метод добавляет элемент, переданный в качестве аргумента, в конец списка. Например, чтобы добавить элемент *item* в конец списка *list*, используйте следующий код:

```
list.append(item)
```

Загрузите в массив *X* вопросы (слова и словосочетания), представляющие собой значения ключа «Вопрос», а в массив *y* – метки, соответствующие значениям ключа «Студенты».

```
# установление меток, соответствующих намерениям пользователя, для каждого вопроса
X = []
y = []
for key, value in students_data["Студенты"].items():
    for question in value["Вопрос"]:
        X.append(question)
        y.append(key)
print(X)
print(y)
```

В приведённом примере пользователь может интересоваться либо фамилиями студентов, либо изучаемыми ими дисциплинами. В результате каждому вопросу будет присвоена одна из меток: «Фамилия» или «Дисциплины».

Чтобы загрузить вопросы пользователя из файла *json* в массив *X*, а соответствующие метки – в массив *y*, используется циклическая конструкция.

В результате в массивы *X* и *y* были успешно загружены вопросы и соответствующие им метки, которые будут использованы для обучения модели. Процесс обучения сводится к решению задачи классификации, где обученная модель сможет определять метку для заданного вопроса пользователя и, основываясь на этой метке, выбирать соответствующий ответ.

5. Преобразование текста в числовой вектор.

Работа с текстом – одно из ключевых направлений применения алгоритмов машинного обучения. Однако текстовые данные не могут быть напрямую переданы в алгоритмы машинного обучения, так как они работают только с числовыми векторами фиксированного размера, а тексты имеют переменную длину.

Поэтому, прежде чем использовать массив *X* для машинного обучения, необходимо преобразовать содержащиеся в нем слова и фразы в числовые векторы.

Для этого можно воспользоваться модулем *sklearn.feature_extraction.text* из библиотеки *sklearn*. Этот модуль содержит функции, которые позволяют преобразовывать текстовые данные в формат, совместимый с алгоритмами машинного обучения.

Одной из таких функций является *CountVectorizer*(). Она выполняет следующие задачи:

– разделяет текст на отдельные слова (токенизация), используя пробелы и знаки препинания в качестве разделителей токенов;

– подсчитывает количество уникальных слов в каждом документе.

При этом положение слов в тексте не учитывается.

Используйте модуль *CountVectorizer* из библиотеки *sklearn*:

```
# преобразуем текст в числовой вектор
import sklearn
from sklearn.feature_extraction.text import CountVectorizer
students_vectorizer = CountVectorizer()
students_m = students_vectorizer.fit_transform(X)
students_m.toarray()
```

6. Обучение модели.

Выберите метод классификации *RandomForestClassifier*() из модуля *sklearn.ensemble*.

Основой метода *RandomForestClassifier*() является другой метод – деревья решений (*decision trees*).

Метод деревьев решений работает следующим образом: в каждом узле проверяется определённый признак объекта. На основе проверки выполняется переход по соответствующей ветке. Процесс повторяется до тех пор, пока не будет

достигнут лист, содержащий информацию о принадлежности объекта к определённому классу.

На рис. 1 представлен пример классификации потенциальных заёмщиков на два класса: «Клиенты, которым банк одобрил кредит» и «Клиенты, которым банк не одобрил кредит».

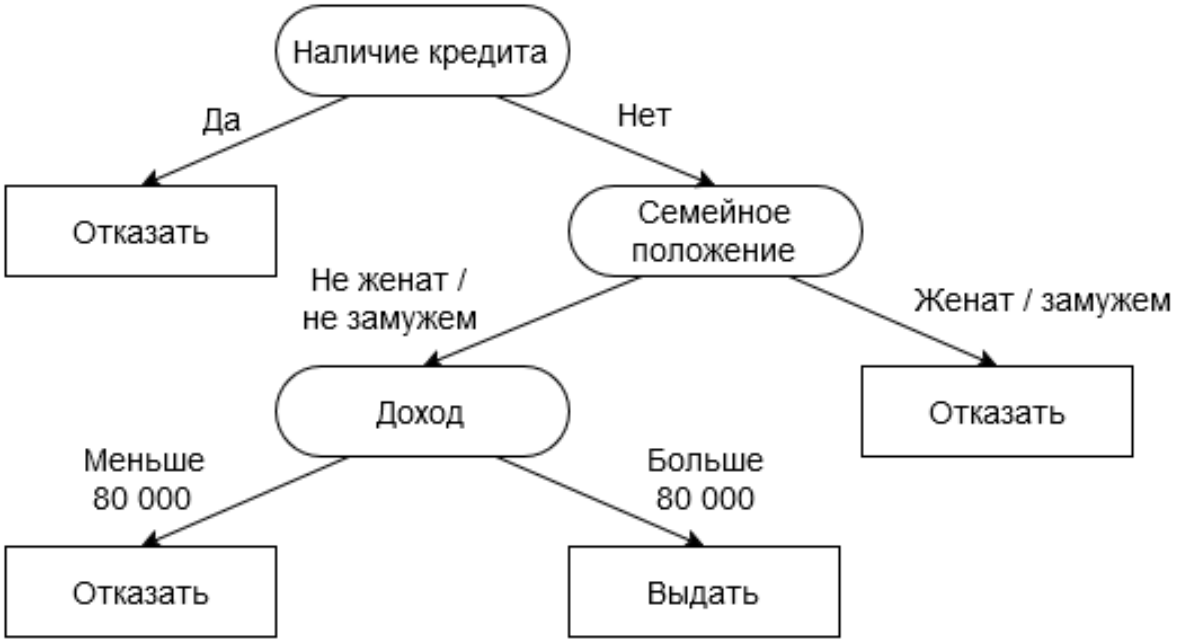

Рис. 1. Пример классификации заемщиков методом *decision trees*

В методе *RandomForestClassifier*() базовые модели представляют собой деревья решений (*decision trees*), которые обучаются на данных, сформированных из случайных подмножеств исходной обучающей выборки. Для повышения точности классификации предсказания этих моделей усредняются.

Импортируйте и настройте метод классификации *RandomForestClassifier*:

```
# загрузка метода классификации RandomForestClassifier() и обучение модели классификации текста
from sklearn.ensemble import RandomForestClassifier
students_RandomFCl = RandomForestClassifier()
students_RandomFCl.fit(students_m, y)
```

7. Проверка модели.

Проверьте работу модели, задав произвольный вопрос.

Для этого введите произвольный вопрос о фамилиях студентов (например, «Как фамилия студента?») или об изучаемых ими дисциплинах (например, «Какие дисциплины вы изучаете?»). Преобразуйте текст вопроса в числовой вектор и проверьте, правильно ли модель присвоила ему метку («Фамилия» или «Дисциплины»), используя функцию *predict*.

```
# проверим качество модели
text = input()
test = students_vectorizer.transform([text])
vopros = students_RandomFCl.predict(test)[0]
print("Метка вопроса: ", vopros)
```

8. Генерация ответа.

Создайте функцию для получения ответа.

Так как ответ генерируется случайным образом из множества возможных вариантов, соответствующих указанному намерению, необходимо предварительно импортировать модуль библиотеки *random*.

```
# создадим функцию, генерирующую ответ пользователю
import random
def get_answer(vopros):
    responses = students_data["Студенты"][vopros]["Ответ"]
    return random.choice(responses)
answer = get_answer(vopros)
print(f"Ответ: {answer}")
```

9. Создание чат-бота.

Создайте чат-бота, который будет отвечать на вопросы пользователя о научном конгрессе «Интерэкспо ГЕО-Сибирь».

9.1. Скачайте файл *GEO-Siberia.json* по следующей ссылке: https://drive.google.com/file/d/1Q7ZvXsw5qONU6zCVqjW4Iam9hGJYHQ-k.

9.2. Данный файл содержит вопросы о научном конгрессе «Интерэкспо ГЕО-Сибирь» и ответы на них, распределенные по следующим разделам:

– *hello* – приветствие;

– *name* – информация о названии;

– *bye* – прощание;

– тематика – темы, которые рассматриваются на конгрессе;

– спикеры – список выступающих;

– длительность – продолжительность мероприятия;

– секции – информация о секциях конгресса;

– цель – цели мероприятия;

– польза – преимущества участия;

– проезд – информация о транспортной доступности;

– проживание – варианты проживания;

– открытие – дата и время открытия;

– выставка – информация о выставках;

– публикация – сведения о публикациях участников;

– награждение – информация о наградах.

В качестве основы для создания чат-бота, отвечающего на вопросы о научном конгрессе «Интерэкспо ГЕО-Сибирь» вы можете использовать программный код:

```
from google.colab import drive
import json
import random
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
# Подключение Google Диска
drive.mount('/content/drive')
# Открытие файла
```

```
smirnov_Geo = open('/content/drive/MyDrive/Colab Notebooks/GEO-Siberia.json',
"r")
Geo = smirnov_Geo.read()
# Загрузка данных из JSON
Geo_test = json.loads(Geo)
# Подготовка данных
XX = []
yy = []
for name, data in Geo_test["Интерэкспо ГЕО-Сибирь"].items():
    for example in data['Вопрос']:
        XX.append(example)
        yy.append(name)
# Преобразование текста в числовые векторы
Geo_vectorizer = CountVectorizer()
Geo_m = Geo_vectorizer.fit_transform(XX)
# Создание и обучение классификатора
Geo_RandomFCl = RandomForestClassifier()
Geo_RandomFCl.fit(Geo_m, yy)
# Функция для получения ответа
def getAnswerGeo(voprosGeo):
    responsesGeo = Geo_test["Интерэкспо ГЕО-Сибирь"][voprosGeo]["Ответ"]
    return random.choice(responsesGeo)
# Основной цикл работы бота
while True:
    textGeo = input("Введите вопрос: ")
    testGeo = Geo_vectorizer.transform([textGeo])
```

```
voprosGeo = Geo_RandomFCl.predict(testGeo)[0]
otvetGeo = getAnswerGeo(voprosGeo)
print("Ответ:", otvetGeo)
```

10. Сохранение работы.

Сохраните блокнот через меню «Файл / Сохранить».

Переименуйте файл на *Google* Диске, используя шаблон «Лабораторная работа № 6 студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

Задание

Создайте чат-бота, который будет отвечать на вопросы пользователей в соответствии с темой (табл. 1).

Бот должен обрабатывать вопросы по пяти разделам. Для каждого раздела необходимо составить:

– 10 вопросов-синонимов, которые пользователь может задать;

– 5 ответов-синонимов, которые бот может предоставить.

Табл. 1. Варианты выполнения задания

Первая буква фамилии студента	Тема	Первая буква фамилии студента	Тема
А	Любимый фильм	О	Любимое время суток
Б	Любимая книга	П	Любимый цвет
В	Любимый город	Р	Любимый цветок
Г	Любимый вид спорта	С	Любимый мультфильм
Д	Любимое время года	Т	Любимая компьютерная игра
Е	Любимая страна	У	Любимый праздник

Ё	Любимое блюдо	Ф	Искусственный интеллект
Ж	Любимый вид отдыха	Х	Чат-боты
З	Любимый жанр музыки	Ц	Роботы
И	Любимая музыкальная группа	Ч	Умные гаджеты
Й	Любимый актер	Ш	Учеба в вузе
К	Любимый поэт	Щ	Знаки зодиака
Л	Любимый писатель	Э	Карьера
М	Хобби	Ю	Семья
Н	Любимая учебная дисциплина	Я	Каникулы

Контрольные вопросы

1. Какие платформы и инструменты используются для выполнения лабораторной работы?
2. Что такое *JSON* и для чего он используется?
3. Как с помощью *Python* загрузить данные из *JSON*-файла и преобразовать их в словари?
4. Какова роль метода *RandomForestClassifier* в разработке чат-бота?
5. Какие данные передаются в массивы *X* и *y* для обучения модели?
6. Как проверить качество работы модели с помощью пользовательских вопросов?
7. Какие этапы включает процесс обучения модели на основе текстовых данных?
8. Почему для классификации текста используются деревья решений и случайные леса?
9. Как чат-бот определяет метку для заданного вопроса?
10. Как генерируется ответ пользователю на основе обученной модели?

Содержание отчета

1. Титульный лист

2. Цель работы.

3. Формулировка задания.

4. Описание результатов выполнения задания в текстовом и графическом виде (скриншоты).

5. Ответы на контрольные вопросы.

6. Подробный вывод о проделанной работе.

Лабораторная работа № 7
РАЗРАБОТКА ЧАТ-БОТА НА ОСНОВЕ БИБЛИОТЕК ДЛЯ ОБРАБОТКИ ЕСТЕСТВЕННОГО ЯЗЫКА

Время выполнения – 8 часов (аудиторная работа – 2 часа, самостоятельная работа – 6 часов).

Цель работы: овладеть навыками разработки чат-бота с использованием библиотек для обработки естественного языка (*NLTK*, *re* и др.).

Задачи работы

1. Изучить модули и функции библиотек, предназначенных для обработки естественного языка (например, *NLTK*, *re* и других).

2. Ознакомиться с возможностями *Python* как инструмента для анализа и обработки текстовых данных.

3. Освоить алгоритм объединения нескольких чат-ботов в один универсальный чат-бот.

Перечень обеспечивающих средств

1. Платформа *Google Colab*.

2. Библиотеки и модули *Python*, используемые для обработки естественного языка (например, *NLTK*, *re* и др.).

Общие теоретические сведения

Обработка естественного языка (*NLP*) – это область искусственного интеллекта и лингвистики, которая занимается анализом, пониманием и созданием человеческого языка с помощью компьютеров. Python является одним из наиболее популярных языков программирования для работы с *NLP* благодаря обширному набору библиотек.

Обработка естественного языка становится все более значимой, поскольку она находит применение в различных сферах, таких как:

- автоматический перевод,
- извлечение информации,
- создание вопросно-ответных систем и многое другое.

Точное понимание текста, извлечение скрытых смыслов и генерация «человекопонимаемого» контента возможны только при наличии у разработчиков глубоких знаний инструментов машинной обработки естественного языка.

1. Создание нового блокнота.

В новом блокноте создайте текстовую ячейку и впишите следующий текст: «Лабораторная работа студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

2. Создание чат-бота на основе библиотек *NLP*.

Наиболее популярной библиотекой с богатым инструментарием для обработки естественного языка считается библиотека *Natural Language Toolkit* (*NLTK*).

Установите библиотеку *Natural Language Toolkit* (*NLTK*) и загрузите необходимые модули.

```
# загружаем библиотеку NLTK
!pip install nltk
import nltk
from nltk.tokenize import word_tokenize
# загрузка модуля punkt
nltk.download('punkt')
```

3. Токенизация текста.

Используйте функцию *word_tokenize* для разбивки предложения на слова.

```
# токенизируем предложение
text = input("Введите предложение для токенизации: ")
word_tokens = word_tokenize(text)
```

```
print("Первоначальный текст:", text)
print("Токенизированный текст:", word_tokens)
```

4. Сравнение строк с использованием *edit_distance*.

Функция *edit_distance* сравнивает две строки и подсчитывает количество символов, в которых они отличаются.

Общение пользователя с чат-ботом предполагает, что человек может варьировать отдельные слова в вопросе, сохраняя его общий смысл. Кроме того, пользователь может допускать ошибки при написании отдельных слов.

Поэтому разработчик чат-бота должен предусмотреть определённую степень гибкости в распознавании формулировок, задаваемых пользователем.

После ввода вопроса пользователем необходимо сравнить его с вопросом, заданным в программном коде. Если различия между ними незначительны, вопрос пользователя может быть идентифицирован как синоним программного вопроса.

Для сравнения вопросов используйте функцию *edit_distance*(*text*1, *text*2), где *text*1 – это вопрос, заданный пользователем, а *text*2 – это вопрос, прописанный в коде (например, «Какая сейчас погода?»).

Загрузите функцию *edit_distance* из библиотеки *NLTK* для сравнения строк.

```
# сравниваем вопрос пользователя с вопросом, сохраненным в памяти программы
text1 = input("Введите вопрос пользователя: ")
reference_question = "Какая сейчас погода?"
distance = nltk.edit_distance(text1, reference_question)
print("Вопросы различаются на", distance, "символов")
```

5. Расчет процента различий между строками.

Подсчитывать абсолютное количество различий между символами в двух строках без учета их длины не совсем корректно. Чем длиннее строки, тем больше допустимых различий между ними.

Функция *get_rank*(*text*1) должна вычислять процент несовпадающих символов относительно средней длины этих строк.

```
def get_rank(text1):
    reference_question = "Какая сейчас погода?"
    distance = nltk.edit_distance(text1, reference_question)
    average_length = (len(text1) + len(reference_question)) / 2
    return (distance / average_length) * 100
text1 = input("Введите вопрос пользователя: ")
print("Различия составляют", get_rank(text1), "процентов")
```

Установите процентное различие между вопросом пользователя и заранее заданным вопросом, например, на уровне 40 %. Если различие между ними составляет меньше указанного процента, программа распознаёт вопрос пользователя как синонимичный программному вопросу. В противном случае вопрос пользователя не будет распознан.

```
# идентификация вопроса пользователя
if get_rank(text1) < 40:
    print("Расхождение между заданным пользователем вопросом и вопросом 'Какая сейчас погода?' составляет менее 40 %, поэтому вопрос идентифицирован")
elif get_rank(text1) < 100:
    print("Расхождение между заданным пользователем вопросом и вопросом 'Какая сейчас погода?' составляет более 40 %, поэтому вопрос не идентифицирован")
else:
    print("Вопрос пользователя и вопрос 'Какая сейчас погода?' абсолютно не совпадают")
```

6. Использование модуля регулярных выражений *re*.

Регулярные выражения позволяют искать, заменять и разбивать текст на основе шаблонов. Одна из наиболее популярных функций модуля *re* – функция *search*(), которая используется для поиска шаблона в строке. Если совпадение найдено, функция возвращает объект совпадения, иначе – *None*.

Найдите слово «погода» в тексте с помощью функции *search*().

```
# найдем шаблон в строке текста
import re
text = input("Введите текст. Заданный шаблон поиска – слово 'погода': ")
pattern = "погода"
poisk = re.search(pattern, text)
if poisk:
    print("Совпадение найдено:", poisk.group())
else:
    print("Совпадение не найдено")
```

7. Использование функции *lower*() для преобразования текста в нижний регистр.

Функция *lower*() преобразует прописные буквы в строчные. Это полезно, так как пользователь может вводить текст в произвольном регистре.

```
# преобразуем прописные буквы в строчные буквы с помощью функции lower()
text_propisnie = input("Введите текст прописными буквами: ")
text_strochnie = text_propisnie.lower()
print("Введенный текст:", text_propisnie)
print("Преобразованный текст:", text_strochnie)
```

8. Использование функции *sub*() для удаления знаков препинания.

Данная функция ищет в тексте *string* шаблон *pattern*, который нужно найти, и заменяет его на выражение *repl*.

При общении с чат-ботом пользователь может вводить вопросы, содержащие знаки препинания. Эти знаки не влияют на смысл вопроса, поэтому их необходимо исключить из текста перед обработкой.

Для исключения знаков препинания создайте шаблон, включающий все такие символы: *punctuation = r*"[^\w\s]", где:

- ^ – обозначает отрицание последовательности в шаблоне;
- \ – указывает, что следующий символ является специальным символом в регулярном выражении;
- \w – соответствует всем буквам, цифрам и символу подчеркивания;
- \s – соответствует любым пробельным символам;
- ^\w\s – исключает любые символы, которые не являются буквами, цифрами, символом подчеркивания или пробелами.

Исключите все знаки препинания в вопросе пользователя.

```
# исключим все знаки препинания
import re
text = input("Введите текст со знаками препинания: ")
punctuation = r"[^\w\s]"
print(re.sub(punctuation, "", text))
```

9. Создание функции *normalize*() для нормализации текста.

В предыдущих пунктах мы использовали функцию *lower*() для преобразования текста в нижний регистр и функцию *sub*() для удаления из исходного текста знаков препинания. Теперь объединим их в одну функцию *normalize*, которая преобразует текст в нижний регистр и удалит знаки препинания.

Создайте пользовательскую функцию *normalize* для нормализации вопроса пользователя.

```
# преобразуем текст в нижний регистр
import re
def normalize(text):
    text = text.lower()
```

```
    punctuation = r"[^\w\s]"
    return re.sub(punctuation, "", text)
```

10. Создание функции *get_rank_normalize* для вычисления различий между текстами.

Создайте функцию *get_rank_normalize*, которая подсчитает, на сколько процентов различаются нормализованный вопрос пользователя и вопрос, записанный в программе.

```
# создаем функцию get_rank_normalize
import nltk
def get_rank_normalize(text1, text2):
    text1 = normalize(text1)
    text2 = normalize(text2)
    distance = nltk.edit_distance(text1, text2)
    average_length = (len(text1) + len(text2)) / 2
    return (distance / average_length) * 100
```

11. Загрузка файла с *Google* Диска в *Google Colab*.

Подключите *Google* Диск к среде *Google Colab* и загрузите файл с информацией о научном конгрессе «Интерэкспо ГЕО-Сибирь» (см. лабораторную работу № 6), используя *Google* Диск в среде *Google Colab*.

```
# загружаем файл с информацией о научном конгрессе «Интерэкспо ГЕО-Сибирь» в среду Google Colab
from google.colab import drive
drive.mount('/content/drive')
```

После подключения *Google* Диска к среде *Google Colab* найдите на подключенном *Google* Диске файл с информацией о научном конгрессе «Интерэкспо ГЕО-Сибирь» и скопируйте путь к этому файлу.

```
import json
smirnov_Geo = open('/content/drive/MyDrive/GEO-Siberia.json', "r")
```

```
Geo = smirnov_Geo.read()
Geo_test = json.loads(Geo)
```

12. Создание функции *getIntent*() для определения объекта.

Введите две новые переменные:

- *best_rank* с начальным значением 70;
- *result* с начальным значением *None*.

Значение 70 у переменной *best_rank* предполагает, что функция *get_rank_normalize* возвращает числовой результат, отражающий степень различия между пользовательским вопросом и вопросами из базы данных. Чем меньше это значение, тем ближе вопросы друг к другу. Значение 70 в нашем случае выбрано эмпирическим путём как максимальный допустимый порог различий.

Загрузите метод *items*(), который возвращает копию списка пар «ключ-значение» словаря.

Ключом (*name*) должны быть названия разделов информации о научном конгрессе «Интерэкспо ГЕО-Сибирь»: «Тематика», «Спикеры», «Длительность», «Секции» и др. Значениями (*data*) должны быть другие объекты, включенные в разделы информации о конгрессе.

Используйте два цикла *for*.

Назначение этих циклов – найти значение ключа «Вопрос» в загруженном файле *GEO-Siberia.json*, для которого значение функции *get_rank_normalize* будет минимальным. Другими словами, нужно найти в файле *GEO-Siberia.json* вопрос, с наименьшими отличиями в символах от пользовательского вопроса.

```
# создаем функцию getIntent(text)
text = input("Задайте вопрос про научный конгресс «Интерэкспо ГЕО-Сибирь»
")
def getIntent(text):
    Geo_dis = Geo_test["Интерэкспо ГЕО-Сибирь"]
    best_rank = 70
```

```
    result = None
    for name, data in Geo_dis.items():
        for question in data["Вопрос"]:
            rank = get_rank_normalize(text, question)
            if rank < best_rank:
                best_rank = rank
                result = name
    return result
```

13. Вывод результатов функции *getIntent*.

Выведите результаты функции *getIntent* на экран.

```
# проверим функцию getIntent(text)
import random
namerenie = getIntent(text)
if namerenie:
    print("Про что пользователь задал вопрос:", namerenie)
else:
    failure_phrases = ("Пожалуйста, перефразируйте вопрос", "Вы спрашиваете не о конференции")
    print(random.choice(failure_phrases))
```

14. Генерация ответа на вопрос пользователя.

Поиск ответа осуществляется в файле *GEO-Siberia.json*, который предварительно загружен в среду *Google Colab*. Ответ находится в объекте «Ответ», относящемся к разделу информации о конференции. Название раздела определяется с помощью функции *getIntent*.

Ответ для пользователя выбирается случайным образом из множества возможных значений ключа «Ответ» с использованием функции *choice*() из модуля

random. Эта функция позволяет получить случайный элемент из переданной ей последовательности строк.

Сгенерируйте ответ на вопрос пользователя

```
# сгенерируем ответ на вопрос пользователя
responses = Geo_test["Интерэкспо ГЕО-Сибирь"][namerenie]["Ответ"]
otvet = random.choice(responses)
print(otvet)
```

15. Добавление программного кода чат-бота на основе машинного обучения.

В новой кодовой ячейке напишите программный код чат-бота, основанный на алгоритмах машинного обучения (лабораторная работа № 6).

```
# чат-бот, основанный на алгоритмах машинного обучения
import sklearn
import json
import random
from google.colab import drive
# Загрузка данных
drive.mount('/content/drive')
with open('/content/drive/MyDrive/GEO-Siberia.json', "r") as smirnov_Geo:
    Geo_test = json.load(smirnov_Geo)
XX, yy = [], []
for name, data in Geo_test["Интерэкспо ГЕО-Сибирь"].items():
    for example in data['Вопрос']:
        XX.append(example)
        yy.append(name)
# Подготовка модели
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.ensemble import RandomForestClassifier
Geo_vectorizer = CountVectorizer()
Geo_m = Geo_vectorizer.fit_transform(XX)
Geo_RandomFCl = RandomForestClassifier()
Geo_RandomFCl.fit(Geo_m, yy)
# Предсказание ответа
textGeo = input("Введите ваш вопрос: ")
testGeo = Geo_vectorizer.transform([textGeo])
voprosGeo = Geo_RandomFCl.predict(testGeo)[0]
def getAnswerGeo(voprosGeo):
    responsesGeo = Geo_test["Интерэкспо ГЕО-Сибирь"][voprosGeo]["Ответ"]
    return random.choice(responsesGeo)
otvetGeo = getAnswerGeo(voprosGeo)
print(otvetGeo)
```

16. Логика работы чат-бота.

Сначала выполняется программный код чат-бота, использующего библиотеки обработки естественного языка. Если этот код не находит соответствующий раздел информации о научном конгрессе «Интерэкспо ГЕО-Сибирь» для заданного пользователем вопроса, то поиск продолжается с использованием чат-бота, работающего на алгоритмах машинного обучения.

```
# объединим два чат-бота
intent = getIntent(text)
if intent:
    responses = Geo_test["Интерэкспо ГЕО-Сибирь"][intent]["Ответ"]
    otvet = random.choice(responses)
    print(otvet)
else:
```

```
text1 = Geo_vectorizer.transform([text])
intent = Geo_RandomFCl.predict(text1)[0]
responses = Geo_test["Интерэкспо ГЕО-Сибирь"][intent]["Ответ"]
otvet = random.choice(responses)
print(otvet)
```

17. Сохранение работы.

Сохраните блокнот через меню «Файл / Сохранить».

Переименуйте файл на *Google* Диске, используя шаблон «Лабораторная работа № 7 студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

Задание

Создайте чат-бота, который использует библиотеки обработки естественного языка для ответа на вопросы пользователей по выбранной теме в рамках лабораторной работы № 6.

Объедините его с чат-ботом, разработанным в ходе выполнения лабораторной работы № 6, чтобы создать единую систему.

Контрольные вопросы

1. Что такое обработка естественного языка (*NLP*), и какие задачи она решает?
2. Какие библиотеки *Python* используются для обработки естественного языка?
3. Какие функции библиотеки *NLTK* применяются в лабораторной работе?
4. Как осуществляется токенизация текста в *NLTK*?
5. Как работает функция *edit_distance*, и для чего она используется в чат-боте?
6. Почему важно учитывать процент различий между строками при сравнении вопросов?

7. Как можно использовать регулярные выражения для обработки пользовательских запросов?

8. Как преобразовать текст в нижний регистр с помощью *Python*?

9. Для чего применяется функция *sub*() в модуле *re*?

10. Что делает функция *normalize*(), и почему она важна при обработке текста?

11. Как работает функция *get_rank_normalize*() для вычисления различий между текстами?

12. Как работает функция *getIntent*(), и какую роль она выполняет в чат-боте?

Содержание отчета

1. Титульный лист

2. Цель работы.

3. Формулировка задания.

4. Описание результатов выполнения задания в текстовом и графическом виде (скриншоты).

5. Ответы на контрольные вопросы.

6. Подробный вывод о проделанной работе.

Лабораторная работа № 8

ЛАБОРАТОРНАЯ РАБОТА № 8. РАЗРАБОТКА TELEGRAM-БОТА НА PYTHON

Время выполнения – 8 часов (аудиторная работа – 2 часа, самостоятельная работа – 6 часов).

Цель работы: овладеть навыками разработки *Telegram*-бота на *Python* с использованием *API Telegram*.

Задачи работы

1. Освоить работу с *API Telegram*.
2. Изучить библиотеку *python-telegram-bot*, её пакеты, методы *API*, обработчики команд и сообщений, а также фильтры обработчиков.
3. Разработать *Telegram*-бота, отвечающего на вопросы пользователей по заданной тематике в рамках итогового задания по учебной дисциплине «Системы искусственного интеллекта».

Перечень обеспечивающих средств

1. Платформа *Google Colab*.
2. Мессенджер *Telegram*.

Общие теоретические сведения

Telegram – это бесплатный мессенджер для обмена сообщениями, изображениями и документами различных форматов (*XLS, PDF*, *DOCX* и др.).

API (*Application Programming Interface*) – программный интерфейс, позволяющий одной программе взаимодействовать с другой с помощью набора методов и инструментов.

Telegram API – программный интерфейс, через который *Telegram* интегрируется с внешними сервисами.

Telegram Bot API – программный интерфейс, используемый для создания ботов в *Telegram*.

1. Создание *Telegram*-бота.

Для создания телеграм-бота воспользуйтесь сервисом *BotFather*.

BotFather – это официальный бот в *Telegram*, предназначенный для регистрации и управления пользовательскими ботами.

Ниже перечислим шаги по созданию бота.

Откройте приложение *Telegram* и введите в строке поиска *@BotFather*. Убедитесь, что у бота есть синяя галочка.

Начните взаимодействие с ботом, отправив команду /start

Введите команду /newbot для создания нового бота.

Укажите уникальное имя бота (оно будет отображаться в верхней строке чата).

Укажите юзернейм бота (он должен быть на английском языке, содержать только буквы и цифры и заканчиваться на *bot*, например, *Familiya*01052025_*bot*, где *Familiya* – фамилия студента на английском языке, 01052025 – дата создания бота в формате ДДММГГГ (день, месяц, год).

BotFather отправит вам токен – уникальный ключ, который потребуется для управления ботом. Сохраните токен в файле *token.txt*. Загрузите этот файл в своем *Google* Диске.

Введите команду /*mybots*, выберите бота и перейдите в *Edit Bot* / *Edit Description*. Укажите описание бота (оно будет отображаться пользователям при открытии чата).

2. Создание нового блокнота.

В новом блокноте создайте текстовую ячейку и впишите следующий текст: «Лабораторная работа студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

Установите последнюю версию библиотеки *python-telegram-bot*:

```
!pip install python-telegram-bot --upgrade
```

3. Установка необходимых библиотек.

```
# импортируем библиотеки
import nltk
import re
import random
import json
import asyncio
import nest_asyncio
from google.colab import drive
from telegram import Update
from telegram.ext import Application, CommandHandler, MessageHandler, filters,
CallbackContext
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
```

4. Подключение *Google* Диска, загрузка токена и *json*-файла.

Код, расположенный ниже, написанный через пустую строку следует размещать в разных ячейках.

Вместо переменной *Smirnov* укажите переменную с вашей фамилией.

```
# подключаем Google Диск для загрузки токена
drive.mount('/content/drive')


# задаем путь к файлу с токеном бота
TOKEN_PATH = "/content/drive/MyDrive/token.txt"
try:
    with open(TOKEN_PATH, "r") as token_file:
        TOKEN = token_file.read().strip()
        if not TOKEN:
```

```
            raise ValueError("Файл токена пуст.")
except (FileNotFoundError, ValueError) as e:
    print(f"Ошибка загрузки токена: {e}")
    TOKEN = None  # Если токен не найден, бот не запустится

# загружаем JSON с вопросами и ответами
JSON_PATH = "/content/drive/MyDrive/GEO-Siberia.json"
try:
    with open(JSON_PATH, "r", encoding="utf-8") as file:
        Smirnov = json.load(file)
except (FileNotFoundError, json.JSONDecodeError) as e:
    print(f"Ошибка загрузки JSON-файла: {e}")
    Smirnov = {}

# проверяем наличие ключевого раздела в JSON
EXPO_KEY = "Интерэкспо ГЕО-Сибирь"
if EXPO_KEY not in Smirnov:
    print(f"Предупреждение: ключ '{EXPO_KEY}' не найден в JSON.")
    Smirnov[EXPO_KEY] = {}
```

5. Создание объекта приложения *Telegram*-бота.

Проверим, задан ли токен, и либо создадим объект *Telegram*-бота, либо остановим выполнение, если токен отсутствует.

```
# создаем объект Telegram-приложения, если токен загружен
if TOKEN:
    app = Application.builder().token(TOKEN).build()
else:
```

```
    print("Ошибка: бот не может запуститься без токена.")
    exit()
```

6. Формирование обучающих данных (X, y) для модели на основе словаря.

```
# формируем обучающие данные для модели
X, y = [], []
for name, data in Smirnov[EXPO_KEY].items():
    for example in data.get("Вопрос", []):
        X.append(example)
        y.append(name)
```

7. Обработка текста и обучение модели для определения намерения пользователя.

```
# загрузка модуля punkt
nltk.download('punkt')


# проверяем, есть ли данные для обучения
if not X or not y:
    print("Предупреждение: нет данных для обучения модели!")


# векторизация текста
vectorizer = TfidfVectorizer()
XX = vectorizer.fit_transform(X) if X else None


# обучение модели, если есть данные
model = RandomForestClassifier()
if XX is not None and y:
    model.fit(XX, y)
```

```
# функция нормализации текста
def normalize(text):
    return re.sub(r"[^\w\s]", "", text.lower())

# функция оценки схожести строк
def get_rank(text1, text2):
    text1, text2 = normalize(text1), normalize(text2)
    if not text1 or not text2:
        return 100
    return (nltk.edit_distance(text1, text2) / ((len(text1) + len(text2)) / 2)) * 100

# определение намерения пользователя
def get_intent(text):
    best_rank = 50
    result = None
    for name, data in Smirnov[EXPO_KEY].items():
        for question in data.get("Вопрос", []):
            rank = get_rank(text, question)
            if rank < best_rank:
                best_rank, result = rank, name
    return result
```

8. Реализация функции ответа бота на основе правил и машинного обучения.

```
# Функция ответа бота
def bot(text):
    intent = get_intent(text)
```

```
    if intent in Smirnov[EXPO_KEY]:
        return random.choice(Smirnov[EXPO_KEY][intent].get("Ответ", ["Ответ не найден"]))
    if XX is not None:
        test = vectorizer.transform([text])
        probabilities = model.predict_proba(test)
        predicted_intent = model.predict(test)[0]
        confidence = max(probabilities[0])  # Берем максимальную вероятность
        print(f"ML-модель предсказала: {predicted_intent} с уверенностью {confidence:.2f}")
        if confidence < 0.3:  # Устанавливаем порог, например 30%
            return "Извините, я не понимаю ваш вопрос."
        if predicted_intent in Smirnov[EXPO_KEY]:
            return random.choice(Smirnov[EXPO_KEY][predicted_intent].get("Ответ", ["Ответ не найден"]))
    return "Извините, я не понимаю ваш вопрос."
```

9. Обработка команд и сообщений для *Telegram*-бота.

```
# обработчик команды /start
async def start(update: Update, context: CallbackContext):
    await update.message.reply_text("Привет! Я бот, отвечающий на вопросы о научном конгрессе Интерэкспо ГЕО-Сибирь. Задайте мне вопрос!")

# обработчик текстовых сообщений
async def send_message(update: Update, context: CallbackContext):
    user_text = update.message.text
    response = bot(user_text)
```

```
    await update.message.reply_text(response)


# добавляем обработчики сообщений
app.add_handler(CommandHandler("start", start))
app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, send_mes-
sage))
```

10. Запуск бота.

```
# запуск бота
nest_asyncio.apply()
async def main():
    await app.run_polling()
# Проверяем, работает ли код в среде, поддерживающей event loop
if __name__ == "__main__":
    try:
        asyncio.run(main())
    except RuntimeError:
        print("Асинхронный event loop уже запущен")
```

11. Откройте *Telegram*-бот и задайте в чате вопросы о научном конгрессе «Интерэкспо ГЕО-Сибирь».

12. Сохранение работы.

Сохраните блокнот через меню «Файл / Сохранить».

Переименуйте файл на *Google* Диске, используя шаблон «Итоговый проект по созданию телеграм-бота студента группы … Фамилия Имя», указав свой номер группы, фамилию и имя.

Задание

Создайте *Telegram*-бота, использующего библиотеки обработки естественного языка и алгоритмы машинного обучения. Бот должен отвечать на вопросы пользователей по заданной теме в рамках лабораторной работы № 6.

Контрольные вопросы

1. Что такое *API*?
2. Какую функцию выполняет сервис *BotFather*?
3. Какие основные этапы необходимо выполнить для создания *Telegram*-бота?
4. Какие методы машинного обучения используются в итоговом проекте?
5. Как бот выбирает наиболее подходящий ответ для пользователя?
6. Как можно улучшить точность работы бота?

Содержание отчета

1. Титульный лист
2. Цель работы.
3. Формулировка задания.
4. Описание результатов выполнения задания в текстовом и графическом виде (скриншоты).
5. Ответы на контрольные вопросы.
6. Подробный вывод о проделанной работе.