Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА №4**

по дисциплине

‘Системы искусственного интеллекта‘

*Выполнил:*

Студент группы P33312

Соболев Иван Александрович

*Преподаватель:*

Кугаевских Александр Владимирович



Санкт-Петербург, 2023

**Задание:**

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

**Этапы реализации и пояснения:**

На первых шагах были импортированы нужные библиотеки, далее с помощью библиотеки pandas считан датасет и выведена основная статистика по нему.

Дальше необходимо было сделать предварительную обработку и нормировку данных, для этого была написана функция нормирования.

def standardize\_data(data):  
 mean = np.mean(data, axis=0)  
 std = np.std(data, axis=0)  
 standardized\_data = (data - mean) / std  
 return standardized\_data

Создаем функцию standardize\_data, которая нормирует данные, приводя их к единому масштабу. Сначала мы вычисляем среднее значение (mean) и стандартное отклонение (std) для каждого признака. Затем мы вычитаем среднее значение из каждой точки данных и делим на стандартное отклонение, чтобы получить нормированные данные.

Когда данные прошли предварительную обработку их было необходимо разделить на тренировочные и тестовые.

# Определяем зависимую переменную (целевую) и признаки  
X = data.drop(columns=['Performance Index'])  
y = data['Performance Index']  
  
  
# Разделение на обучающий и тестовый наборы  
def train\_test\_split\_custom(X, y, test\_size=0.2):  
 num\_samples = X.shape[0]  
 num\_test\_samples = int(test\_size \* num\_samples)  
  
 # Генерация случайных индексов для тестового набора  
 test\_indices = np.random.choice(num\_samples, num\_test\_samples, replace=False)  
  
 # Индексы для обучающего набора  
 train\_indices = np.setdiff1d(np.arange(num\_samples), test\_indices)  
  
 X\_train, X\_test = X.iloc[train\_indices], X.iloc[test\_indices]  
 y\_train, y\_test = y.iloc[train\_indices], y.iloc[test\_indices]  
  
 return X\_train, X\_test, y\_train, y\_test  
  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split\_custom(X, y, test\_size=0.2)

Создаем функцию train\_test\_split\_custom, которая разделяет данные на обучающий и тестовый наборы. Эта функция случайным образом выбирает индексы для тестового набора данных, исходя из заданного коэффициента test\_size. Таким образом, мы получаем два набора данных: X\_train, y\_train - обучающий набор, и X\_test, y\_test - тестовый набор.

Дальше создаём основной модуль линейной регрессии.

def perform\_linear\_regression(columns, X\_train, X\_test, y\_train, y\_test):  
 # Добавляем столбец с единицами для учета свободного члена  
 X\_train['intercept'] = 1  
 X\_test['intercept'] = 1  
 if columns != None:  
 columns = columns.split(",")  
  
 if len(columns) == 1:  
 X\_train = X\_train[['intercept', columns[0]]]  
 X\_test = X\_test[['intercept', columns[0]]]  
 elif len(columns) == 2:  
 X\_train = X\_train[['intercept', columns[0], columns[1]]]  
 X\_test = X\_test[['intercept', columns[0], columns[1]]]  
 else:  
 X\_train = X\_train[['intercept', columns[0], columns[1], columns[2]]]  
 X\_test = X\_test[['intercept', columns[0], columns[1], columns[2]]]  
 # Преобразуем данные в массивы NumPy  
 X\_train = X\_train.to\_numpy()  
 X\_test = X\_test.to\_numpy()  
 y\_train = y\_train.to\_numpy()  
 y\_test = y\_test.to\_numpy()  
  
 # Вычислим коэффициентов линейной регрессии методом наименьших квадратов  
 coefficients = np.linalg.lstsq(X\_train, y\_train, rcond=None)[0]  
  
 # Вычислим суммы квадратов  
 def sum\_of\_squares(y\_true, y\_pred):  
 return np.sum(np.square(y\_true - y\_pred))  
  
 # Получим предсказания для тестового набора данных  
 y\_pred = np.dot(X\_test, coefficients)  
  
 # Оценка производительности с использованием коэффициента детерминации (R^2)  
 def r2\_score\_custom(y\_true, y\_pred):  
 total\_variance = np.sum((y\_true - np.mean(y\_true)) \*\* 2)  
 residual\_variance = np.sum((y\_true - y\_pred) \*\* 2)  
 r2 = 1 - (residual\_variance / total\_variance)  
 return r2  
  
 # Вычислим R^2 для модели  
 r2 = r2\_score\_custom(y\_test, y\_pred)  
 sum\_of\_squares = sum\_of\_squares(y\_test, y\_pred)  
  
 return y\_pred, r2, sum\_of\_squares

В первой строчке добавляем свободный член, чтобы итоговая прямая не была привязана к началу координат. Далее преобразуем данные в массивы NumPy и затем используем функцию np.linalg.lstsq() для вычисления коэффициентов линейной регрессии методом наименьших квадратов. Результатом является массив coefficients, содержащий коэффициенты регрессии. После этого мы используем вычисленные коэффициенты для предсказания значений целевой переменной (y\_pred) на тестовом наборе данных, умножая матрицу признаков X\_test на вектор коэффициентов с помощью функции np.dot().

Далее просто создаем несколько моделей и анализируем их.

**Выводы:**

Можно заметить, что коэффициент детерминации сильно повысился за счёт Previous Scores и Hours Studied, можем сделать вывод, что успеваемость зависит от имеющихся знаний студента (предыдущих оценок), и также от того, сколько часов он потратил на учебу. Если у студента хорошие знания и мало учился, то скорее успеваемость у него будет немного меньше, а если хорошо подготовился, то будет примерно такой же балл. Если у студента плохие знания и много учился, то у него скорее всего сильно поднимется оценка. Также мотивация немного поднимает коэффициент детерминации, следовательно, чем выше мотивация, тем выше будет успеваемость.