Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Севастопольский государственный университет»

ОТЧЕТ

о выполнении лабораторной работы № 1 по дисциплине

«Методы и системы искусственного интеллекта»

Выполнил:

ст. гр. ИС/б-22-1-о

Гюнтер М. Ю .

Проверил:

доцент кафедры

“Информационные системы”

Бондарев В. Н.

Севастополь,2025

# Цель работы: Изучение технологии подготовки и выполнения программ на языке Python, исследование свойств функций языка Python, используемых при обработке последовательностей, формирование навыков определения классов языка Python.

# Ход работы

## Выполнение задания 1

Продемонстрируем работу метода str.split() (рисунок ‎2.1), возвращающего список на основании строки и разделителя, по умолчанию равного пробелу, с помощью кода, приведённого в листинге ‎2.1.

Листинг ‎2.1 – Демонстрация метода str.split()

nums\_str = '0 1 2 3 4 5 6 7 8 9'

nums\_int = nums\_str.split()

nums\_inc = [int(num) + 1 for num in nums\_int]

for num in nums\_inc:

print(num, end = " ")



Рисунок ‎2.1 – Результат обработки строки с использованием str.split()

Также продемонстрируем работу аналогичного метода str.join() (рисунок ‎2.2), объединяющего итерируемый объект на основании строки соединителя, для которой вызывается метод, с помощью кода, приведённого в листинге ‎2.2.

Листинг ‎2.2 – Демонстрация метода str.join()

nums\_str\_comma = '0, 1, 2, 3, 4, 5, 6, 7, 8, 9'

print(f'Оригинальная строка: {nums\_str\_comma}')

nums\_int = nums\_str\_comma.split(', ')

nums\_str\_space = " ".join(nums\_int)

print('Строка с изменённым разделителем: {}'.format(nums\_str\_space))



Рисунок ‎2.2 – Изменение разделителя строки с использованием str.join()

Продемонстрируем работу метода str.count() (рисунок ‎2.3), возвращающего количество неперекрывающихся повторений заданной подстроки в строке, с помощью кода, приведённого в листинге ‎2.3.

Листинг ‎2.3 – Демонстрация метода str.count()

haystack = 'Хороший код — это читаемый код.'

needle = 'код'

count = haystack.count(needle)

print(f'''Сколько раз слово "{needle}" содержится в предложении "{haystack}"?\n{count}''')



Рисунок ‎2.3 – Результат поиска подстроки с использованием str.count()

Продемонстрируем работу метода str.startswith() (рисунок ‎2.4), возвращающего True, если строка начинается с заданной подстроки, и False, если строка не начинается с заданной подстроки, с помощью кода, приведённого в листинге ‎2.4.

Листинг ‎2.4 – Демонстрация метода str.startswith()

fruits = [

"Apple",

"banana",

"apricot",

"Grape",

"avocado",

"orange",

"kiwi",

"Almond",

"Blueberry",

"artichoke"

]

count = 0

for i in fruits:

if i.lower().startswith('a'):

count += 1

print('''Количество фруктов, название которых начинается с буквы "a": \n%d''' % count)



Рисунок ‎2.4 – Результат поиска фруктов на букву “a” с использованием str.startswith()

Продемонстрируем работу метода str.endswith() (рисунок ‎2.5), возвращающего True, если строка заканчивается заданной подстрокой, и False, если строка не заканчивается заданной подстрокой, с помощью кода, приведённого в листинге ‎2.5.

Листинг ‎2.5 – Демонстрация метода str.endswith()

sentences = [

"Синтаксис Python очень читаемый.",

"Отладка кода требует терпения",

"Алгоритмы — это основа программирования.",

"Компилятор преобразует код в машинный язык",

"Тестирование необходимо для качества продукта.",

"Версионирование кода с помощью Git обязательно",

"Функции помогают избежать повторения кода.",

"Базы данных хранят информацию структурированно",

"Рекурсия может быть сложной для понимания.",

"Веб-разработка включает фронтенд и бэкенд"

]

count = 0

for i in sentences:

if i.endswith('.'):

count += 1

print('Число предложений, где пропущена точка: \n%d' % count)



Рисунок ‎2.5 – Результат поиска предложений, где пропущена точка, с использованием str.endswith()

## Выполнение задания 2

Был проведён дискриминантный анализ тренировочных данных (рисунок ‎2.6) с помощью команд R, приведённых в листинге ‎2.6.

Листинг ‎2.6 – Команды для проведения дискриминантного анализа

dataset.lda <- lda (Dataset.train[, 1:6], Dataset.train[, 7])

dataset.lda

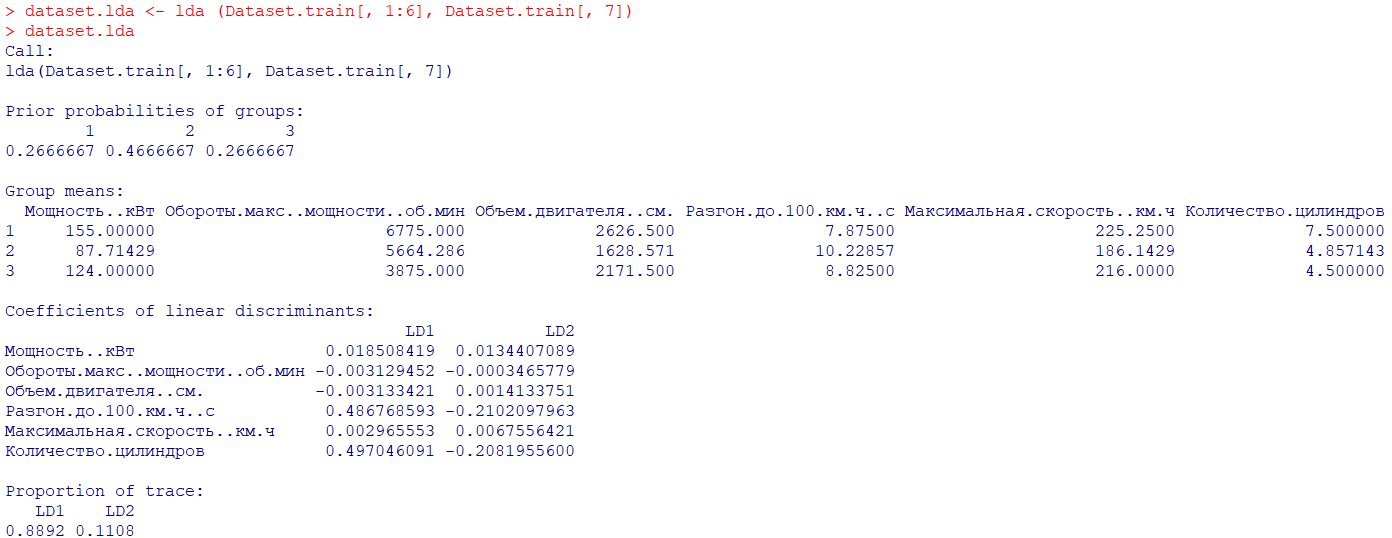


Рисунок ‎2.6 – Результаты проведения дискриминантного анализа тренировочных данных

## Построение дискриминантных функций

В соответствии с результатами дискриминантного анализа были получены коэффициенты линейных дискриминантов (рисунок ‎2.7).

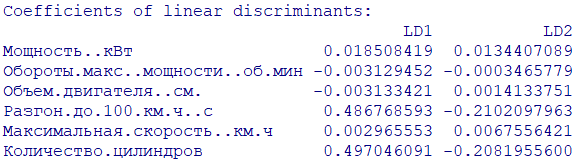


Рисунок ‎2.7 – Коэффициенты линейных дискриминантов

На основании этих коэффициентов построим дискриминантные функции:

z(x) = 0.0185x1 – 0.0031x2 – 0.0031x3 + 0.5x4 + 0.003x5 + 0.5x6

z(x) = 0.0134x1 – 0.0003x2 + 0.0014x3 - 0.21x4 + 0.0068x5 - 0.208x6

## Классификация и проверка оставшихся данных

Была произведена классификация и проверка оставшихся данных (рисунок ‎2.8) с помощью команд R, приведённых в листинге ‎2.7.

Листинг ‎2.7 – Команды для классификации и проверки оставшихся данных

dataset.ldap <- predict(dataset.lda, Dataset.unknow [,1:6])$class

dataset.ldap

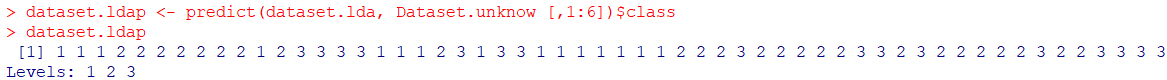


Рисунок ‎2.8 – Результаты классификации и проверки оставшихся данных

Для проверки созданной модели построим матрицу неточностей (рисунок ‎2.9) с помощью команд R, приведённых в листинге ‎2.8.

Листинг ‎2.8 – Команды для построения матрицы неточностей

table (dataset.ldap, Dataset.unknow[,7])

Изображение выглядит как текст, Шрифт, снимок экрана, рукописный текст

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.9 – Результаты построения матрицы неточностей

По данной матрице видно, что тренировочная выборка привела к построению гипотезы, по которой ни один объект не попал не свой класс, следовательно, ошибка в данной модели составляет 0%.

## Проведение шаговой процедуры выбора переменных для построения дискриминантной модели.

Для найденного набора данных была проведена шаговая процедура выбора переменных для построения дискриминантной модели (листинг ‎2.9). В результате выполнения процедуры (рисунок ‎2.10).

Листинг ‎2.9 – Команды для проведения пошаговой процедуры выбора переменных

stepclass(Dataset[,1:6], Dataset[,7], method="lda")

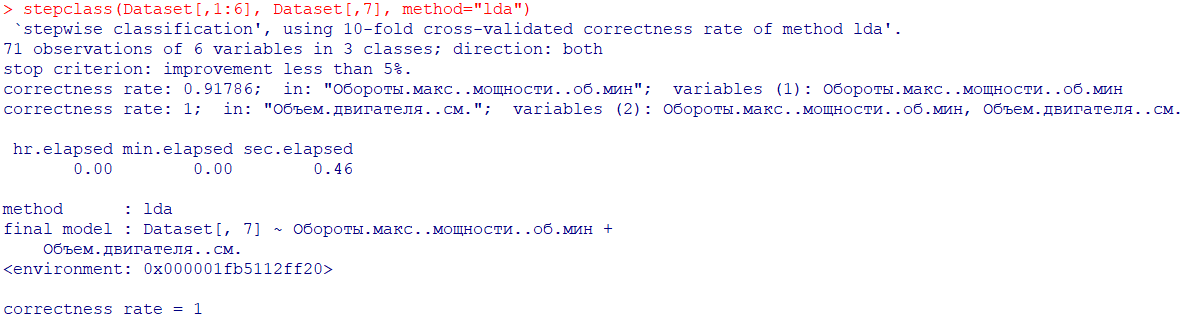


Рисунок ‎2.10 – Результаты проведения пошаговой процедуры выбора переменных

Исходя из результатов выбора переменных для построения дискриминантной модели будут использоваться только две переменные «Обороты максимальной мощности» и «Объем двигателя».

## Построение дискриминантной модели на основании выбранных переменных

Был проведён дискриминантный анализ тренировочных данных с учётом только выбранных на предыдущем этапе независимых переменных (рисунок ‎2.11) с помощью команд R, приведённых в листинге ‎2.10.

Листинг ‎2.10 – Команды для проведения дискриминантного анализа

dataset.lda <- lda (Dataset.train[, 2:3], Dataset.train[, 7])

dataset.lda

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.11 – Результаты проведения дискриминантного анализа тренировочных данных

В соответствии с результатами дискриминантного анализа были получены коэффициенты линейных дискриминантов (рисунок ‎2.12).

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.12 – Коэффициенты линейных дискриминантов

На основании этих коэффициентов построим дискриминантные функции:

z(x) = –0.0025x1 – 0.0019x2

z(x) = 0.0005x1 – 0.0024x2

Была произведена классификация и проверка оставшихся данных с учётом выбранных независимых переменных (рисунок ‎2.13) с помощью команд R, приведённых в листинге ‎2.11.

Листинг ‎2.11 – Команды для классификации и проверки оставшихся данных

dataset.ldap <- predict(dataset.lda, Dataset.unknow [,2:3])$class

dataset.ldap

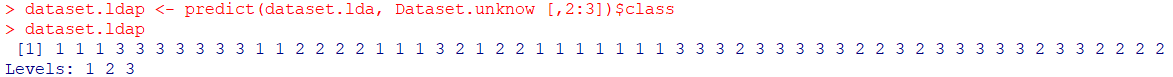


Рисунок ‎2.13 – Результаты классификации и проверки оставшихся данных с учётом выбранных столбцов

Для проверки созданной модели построим матрицу неточностей (рисунок ‎2.14) с помощью команд R, приведённых в листинге ‎2.12.

Листинг ‎2.12 – Команды для построения матрицы неточностей

table (dataset.ldap, Dataset.unknow[,7])

Изображение выглядит как текст, Шрифт, рукописный текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.14 – Результаты построения матрицы неточностей

Также для построенной матрицы посчитаем ошибку распознавания (рисунок ‎2.15) с помощью команд R, приведённых в листинге ‎2.13.

Листинг ‎2.13 – Команды для расчёта ошибки распознавания

Err\_S <- mean (Dataset.unknow[,7] != dataset.ldap)

Err\_S

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.15 – Ошибка распознавания

По результатам подсчёта ошибки распознавания, и в целом по полученной матрице, видно, что тренировочная выборка привела к построению гипотезы, по которой один объект попал не свой класс, следовательно, ошибка в данной модели составляет 0.017857%. Получаем, что ошибка распознавания исходной модели без удаления переменной меньше ошибки распознавания модели с удалёнными переменными.

Теперь рассчитаем расстояние Махалонобиса (рисунок ‎2.16) с помощью команд R, приведённых в листинге ‎2.14.

Листинг ‎2.14 – Команды для расчёта расстояния Махалонобиса

mahDist <- dist(dataset.lda$means %\*% dataset.lda$scaling)

mahDist

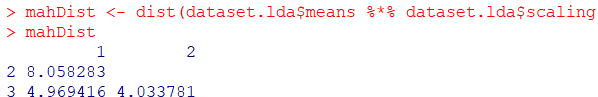


Рисунок ‎2.16 – Рассчитанное расстояние Махалонобиса

## Построение дискриминантной модели на основании выборки с записями без классификации

Добавим в выборку 4 записи без классификации, то есть со значением столбца кластера 0 (рисунок ‎2.17)

Изображение выглядит как текст, снимок экрана, Параллельный, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.17 – Выборка с 4 добавленными записями без классификации

Создадим тренировочную выборку строк от первой до последней проклассифицированной (то есть не включая последние 4 строки) с шагом 5 (рисунок ‎2.18) с помощью команд R, приведённых в листинге ‎2.15.

Листинг ‎2.15 – Команды для создания и выведения на экран тренировочной выборки без последних 4-х строк

Dataset.train <- Dataset [seq (1, nrow(Dataset) - 4, 5),]

Dataset.train

Изображение выглядит как текст, снимок экрана, линия, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.18 – Тренировочная выборка для дискриминантного анализа без не проклассифицированных строк

Аналогично создадим проверочную выборку из оставшихся строк вместе с не проклассифицированными (рисунок ‎2.19) с помощью команд R, приведённых в листинге ‎2.16.

Листинг ‎2.16 – Команды для создания и выведения на экран проверочной выборки вместе с не проклассифицированными строками

Dataset.unknow <- Dataset [-seq (1, nrow(Dataset) - 4, 5),]

Dataset.unknow

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.19 – Конец проверочной выборки для дискриминантного анализа, в которую добавлены не проклассифицированные строки

Был проведён дискриминантный анализ тренировочных данных (рисунок ‎2.20) с помощью команд R, приведённых в листинге ‎2.17.

Листинг ‎2.17 – Команды для проведения дискриминантного анализа

dataset.lda <- lda (Dataset.train[, 1:6], Dataset.train[, 7])

dataset.lda

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.20 – Результаты проведения дискриминантного анализа тренировочных данных

В соответствии с результатами дискриминантного анализа были получены коэффициенты линейных дискриминантов (рисунок ‎2.21).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок ‎2.21 – Коэффициенты линейных дискриминантов

На основании этих коэффициентов построим дискриминантные функции:

z(x) = 0.0185x1 – 0.0031x2 – 0.0031x3 + 0.5x4 + 0.003x5 + 0.5x6

z(x) = – 0.0134x1 + 0.0003x2 – 0.0014x3 + 0.21x4 – 0.0068x5 + 0.208x6

Отметим, что в результате проведения этого дискриминантного анализа, по сравнению с предыдущим случаем, поменялись на противоположные знаки коэффициентов для второй дискриминантной функции.

Была произведена классификация и проверка оставшихся данных с не проклассифицированными строками (рисунок ‎2.22) с помощью команд R, приведённых в листинге ‎2.18.

Листинг ‎2.18 – Команды для классификации и проверки оставшихся данных с не проклассифицированными строками

dataset.ldap <- predict(dataset.lda, Dataset.unknow [,1:6])$class

dataset.ldap

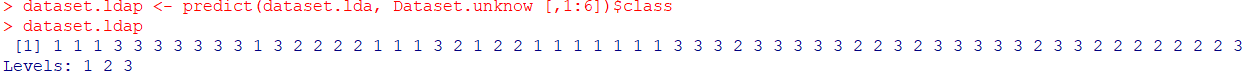


Рисунок ‎2.22 – Результаты классификации и проверки оставшихся данных с не проклассифицированными строками

В результате классификации первые три строки были отнесены ко 2-му кластеру, а последняя к 3-му.

Для проверки созданной модели построим матрицу неточностей (рисунок ‎2.23) с помощью команд R, приведённых в листинге ‎2.19.

Листинг ‎2.19 – Команды для построения матрицы неточностей

table (dataset.ldap, Dataset.unknow[,7])

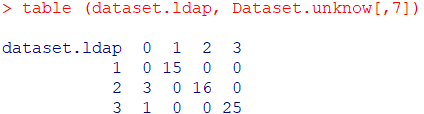


Рисунок ‎2.23 – Результаты построения матрицы неточностей

По построенной матрице убеждаемся в том, что 3 их добавленных строк попали во 2-й кластер, а одна – в 3-й.

# Вывод:

В результате выполнения лабораторной работы был проведён дискриминантный анализ найденных данных при количестве кластеров 3. В результате проведения дискриминантного анализа на тренировочных данных были получены коэффициенты линейных дискриминантов, на основании которых были построены дискриминантные функции. Полученные в ходе дискриминантного анализа данные были проверены на оставшихся данных, и при построении матрицы неточностей выяснилось, что ошибка в построенной модели составляет 0%. Далее была проведена пошаговая процедура выбора переменных для построения дискриминантной модели, в ходе которой было выделено 2 переменные «Обороты максимальной мощности» и «Объем двигателя». При проведении повторного дискриминантного анализа на основе только этих 2-х переменных была получена модель с ошибкой классификации 0.017857%. Также, дополнительно в исходную выборку было добавлено 4 строки с произвольными данными, схожими с таковыми в строках оригинальной выборки. По результатам проведения дискриминантного анализа этой выборки 3 из добавленных строк попали во 2-й кластер, а одна – в 3-й.