Міністерство освіти і науки України
Національний авіаційний університет
Навчально-науковий інститут комп'ютерних інформаційних технологій
Кафедра комп'ютеризованих систем управління

Курсова робота
з дисципліни «Системне програмне забезпечення»

Пояснювальна записка
Тема: реалізація наївного баєсового класифікатора
на мові програмування Python

Виконав:
студент групи СП-325
Клокун В. Д.

Київ — 2018

Завдання на виконання курсової роботи
студента групи СП-325 Клокуна Владислава Денисовича

1. Тема курсової роботи: реалізація наївного баєсового класифікатора на мові програмування Python для класифікації спостережень, що містять неперервні числові дані.

2. Термін виконання курсової роботи:
   з «____» _____ 2018 р. по «____» _____ 2018 р.

3. Вхідні дані до роботи: набір даних для класифікації.

4. Етапи виконання курсової роботи:

   — Огляд теоретичних відомостей про наївний баєсів класифікатор.
   — Постановка задачі.
   — Реалізація наївного баєсового класифікатора.
   — Тестування розробленої реалізації наївного баєсового класифікатора.

5. Перелік обов'язкових додатків і графічного матеріалу:

   — Початковий код розробленої реалізації.
   — Зміст набору вхідних даних «Іриси Фішера».

Завдання отримав: «____» _____ 2018 р.
Підпис студента: _____ (Клокун В. Д.)

Зміст

1. Теоретична частина

1.1. Загальна характеристика наївного баєсового класифікатора

Припустимо, що в ході деякого експерименту проводились спостереження, під час проведення яких збирались неперервні (недискретні) дані про результат події. Також були визначені категорії (або класи), до яких ці дані можуть належати. Поставлена задача класифікувати дані спостережень. *Класифікація* — це задача визначення, до якої з категорій належить певне спостереження. [1] *Класифікатор* — це алгоритм, який виконує класифікацію. [1]

Наївний баєсів класифікатор — це ймовірнісний класифікатор, який використовує теорему Баєса для класифікації спостережень. Такі класифікатори отримують на вхід спостереження, оцінюють його і роблять припущення про клас, до якого воно належить. Вхідні дані, тобто спостереження, представляються у вигляді вектора відомих значень випадкових змінних, які називаються *ознаками*. Результатом роботи класифікатора є певне значення цільової змінної або змінних, які зазвичай називаються класовими, і позначають клас, до якого належить спостереження.

Принцип класифікації полягає в обчисленні умовних імовірностей (визначення 1) того, що вхідні дані належать до певних класів (події, які нас цікавлять), за умови, що ознаки мають певні значення (події, які ми спостерігаємо). Після обчислення кожної з умовних імовірностей за обраним правилом прийняття рішення робиться висновок, до якого класу належить задане спостереження. Оскільки такий класифікатор використовує ймовірнісну модель, наївний баєсів класифікатор називають *імовірнісним*.

Визначення 1 (Умовна ймовірність)**.** Нехай A і B — події. Позначимо ймовірність настання кожної з них незалежно одна від одної як $P(A)$ і $P(B)$ відповідно. Тоді *умовною імовірністю* $P(A \mid B)$ називається ймовірність настання події A за умови, що подія B настала. Вона обчислюється так:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}, \tag{1.1}$$

де $P(A \cap B)$ — ймовірність, що події A і B настали.

Розглянемо приклад класифікації наївним баєсовим класифікатором. Нехай подія A — дане спостереження належить до певного класу, подія B — ознаки спостереження мають певні значення. Тоді щоб знайти ймовірність,

що дане спостереження з певним значенням ознак належить до певного класу, необхідно обчислити умовну ймовірність $P(A \mid B)$. Для обчислення цієї імовірності необхідно використати теорему Баєса (теорема 1).

Теорема 1 (Баєса)**.** Нехай $P(A \mid B)$ — умовна ймовірність настання події A за умови, що подія B настала, $P(B \mid A)$ — умовна ймовірність настання події B за умови, що подія A настала; $P(B)$ — імовірність настання події B, причому $P(B) \neq 0$. Тоді умовна ймовірність $P(A \mid B)$ обчислюється так:

$$P(A \mid B) = \frac{P(B \mid A)\,P(A)}{P(B)}. \tag{1.2}$$

Доведення. Виразимо ймовірності $P(A \mid B)$ і $P(B \mid A)$ за визначенням умовної ймовірності:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)},\ P(B) \neq 0, \tag{1.3}$$

$$P(B \mid A) = \frac{P(B \cap A)}{P(A)},\ P(A) \neq 0. \tag{1.4}$$

Представимо ймовірності настання подій A і B разом, тобто $P(A \cap B)$ і $P(B \cap A)$ з (1.3), (1.4):

$$P(A \cap B) = P(A \mid B)\,P(B), \tag{1.5}$$

$$P(B \cap A) = P(B \mid A)\,P(A). \tag{1.6}$$

Оскільки ліві частини рівні за аксіомою $P(A \cap B) = P(B \cap A)$, то і праві частини також будуть рівними: $P(A \mid B)\,P(B) = P(B \mid A)\,P(A)$, отже можна записати, що:

$$P(A \cap B) = P(B \mid A)\,P(A). \tag{1.7}$$

Підставимо (1.7) в (1.3) і отримаємо:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B \mid A)\,P(A)}{P(B)},\ P(B) \neq 0. \tag{1.8}$$

□

Як видно, при класифікації наївним баєсовим класифікатором, у обчисленні умовних імовірностей використовується теорема Баєса, тому такий імовірнісний класифікатор називається *баєсовим*.

1.2. Імовірнісна модель наївного баєсового класифікатора

Як було сказано у підрозділі 1.1, наївний баєсів класифікатор використовує ймовірнісну модель. Побудуємо та представимо її. Для виконання класифікації необхідні вхідні дані, тобто спостереження, та набір можливих значень класової змінної для позначення класів, до яких можуть належати ці дані. Позначатимемо змінні, на кшталт X_i, великими літерами, а їх значення, наприклад, x_i — малими. Вектори, на зразок $\mathbf{X}$ — жирним шрифтом.

Вхідними даними для класифікації буде вектор ознак $\mathbf{X} = (X_1, \dots, X_n)$, де $X_1, \dots, X_n$ — ознаки. Кожна ознака може мати значення зі своєї області визначення, яка позначається D_i. Набір усіх векторів ознак позначається як $\Omega = D_1 \times \cdots \times D_n$. Для позначення класу, до якого належить спостереження, введемо випадкову змінну C, де C може приймати одне з m значень: $c \in \{0, \dots, m-1\}$.

Визначення 2 (Розподіл імовірностей)**.** Нехай X і Y — випадкові змінні, які приймають значення x та y відповідно. Тоді розподіл імовірностей $p(X \mid Y)$ позначає значення імовірностей $P(X = x_i \mid Y = y_i)$ для кожної з можливих пар i, j. [2]

Класифікація за допомогою наївного баєсового класифікатора ставить у відповідність кожному вектору $\mathbf{X}$, який містить ознаки $X_1, \dots, X_n$, розподіли ймовірностей $p(C \mid \mathbf{X})$. Тобто сама модель має такий загальний вигляд:

$$p(C \mid \mathbf{X}) = p(C \mid X_1, \dots, X_n). \tag{1.9}$$

Зі зростанням кількості або можливих значень ознак, з такою моделлю неможливо працювати за допомогою таблиць імовірностей, тому переформулюємо модель, щоб зробити її зручнішою.

Використовуючи теорему Баєса, представимо її так:

$$p(C \mid X_1, \dots, X_n) = \frac{p(C)\, p(X_1, \dots, X_n \mid C)}{p(X_1, \dots, X_n)}. \tag{1.10}$$

Видно, що дільник не залежить від змінної C, а значення ознак X_i задані наперед, тому на практиці значення дільника постійне. Ділене рівносильне такій моделі спільного розподілу:

$$p(C, X_1, \dots, X_n). \tag{1.11}$$

Перетворюємо дану модель за допомогою визначення умовної ймовірності:

$$p(C, X_1, \dots, X_n) = p(C)\, p(X_1, \dots, X_n \mid C) \tag{1.12}$$
$$= p(C)\, p(X_1 \mid C)\, p(X_2, \dots, X_n \mid C, X_1) \tag{1.13}$$
$$= p(C)\, p(X_1 \mid C)\, p(X_2 \mid C, X_1)\, p(X_3, \dots, X_n \mid C, X_1, X_2) \tag{1.14}$$
$$= p(C)\, p(X_1 \mid C)\, p(X_2 \mid C, X_1)\, p(X_3 \mid C, X_1, X_2)$$
$$p(X_4, \dots, X_n \mid C, X_1, X_2, X_3) \tag{1.15}$$

і так далі. Тепер припускаємо, що кожна ознака X_i умовно незалежна від кожної іншої ознаки X_j, для будь-яких $j \neq i$ та заданої категорії $C = c$. Математично це означає:

$$p(X_i \mid C, X_j) = p(X_i \mid C).$$

Таке припущення є наївним, оскільки немає жодних підстав вважати, що вхідні ознаки дійсно незалежні одна від одної. Саме тому такий баєсів класифікатор називається *наївним*.

Отже, виражаємо загальну модель:

$$p(C \mid X_1, \dots, X_n) = p(C)\, p(X_1 \mid C)\, p(X_2 \mid C) \dots p(X_n \mid C) \tag{1.16}$$
$$= p(C) \prod_{i=1}^{n} p(X_i \mid C). \tag{1.17}$$

Це означає, що враховуючи припущення про незалежність змінних, умовний розподіл над класовою змінною C може бути виражений так:

$$p(C \mid X_1, \dots, X_n) = \frac{1}{Z}\, p(C) \prod_{i=1}^{n} p(X_i \mid C), \tag{1.18}$$

де Z — коефіцієнт, який залежить виключно від $X_1, \dots, X_n$. Якщо значення ознак $x_1, \dots, x_n$ відомі, коефіцієнт Z сталий.

Таку модель значно зручніше використовувати, оскільки вони використовують апріорні імовірності класів $p(C)$ та незалежні розподіли $p(X_i \mid C)$. Якщо є k класів та модель модель для $p(X_i)$ може бути виражена r параметрами, то відповідний наївний баєсів класифікатор матиме $(k-1) + nrk$ параметрів. [3]

1.3. Оцінка параметрів

Описавши ймовірнісну модель, необхідно визначити її параметри, тобто апріорні ймовірності належності до класу $p(C)$ та розподіли ймовірностей

ознак $p(X_i \mid C)$. Для обчислення параметрів моделі використовують *тренувальний набір даних* — такий набір даних, який складається із заздалегідь класифікованих спостережень. Тобто набір даних S складатиметься з векторів $\mathbf{x} = (x_1, \dots, x_n, c)$, де c — правильне значення класової змінної.

Усі параметри моделі можна обчислити з тренувального набору даних. [3] Щоб оцінити значення параметрів, необхідно зробити припущення щодо розподілу, який характеризує дані. Припущення щодо розподілу, який характеризує дані, називають *моделлю подій*. При роботі з неперервними (недискретними) даними, зазвичай припускають, що вони розподілені за законом нормального (гаусового) розподілу. Функція густини ймовірності нормального розподілу така:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{1.19}$$

де μ — математичне сподівання, σ^2 — дисперсія випадкової величини.

Наприклад, припустимо, що тренувальний набір даних містить неперервну ознаку X. Щоб обчислити розподіл імовірності, необхідно спочатку розподілити дані за класами, наданими у тренувальному наборі, та обчислити математичне сподівання μ_c і дисперсію випадкової величини σ_c^2 для кожного з класів. Оскільки набір даних містить не всі можливі значення ознаки, а лише певну вибірку, то її статистичні характеристики — середнє значення вибірки $\overline{x}$ і дисперсія вибірки s^2. Середнє значення вибірки $\overline{x}$ обчислюється як середнє арифметичне значення множини значень ознак: [4]

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}. \tag{1.20}$$

Дисперсія вибірки s^2 обчислюється так: [4]

$$s^2 = \sqrt{\frac{\sum_{i=1}^{n} \left(x_i - \overline{x}\right)^2}{n-1}}. \tag{1.21}$$

Нехай $\overline{x}_{c_1}$ — середнє значення ознак, які належать до класу c_1, а $s_{c_1}^2$ — їх дисперсія. У результаті певного спостереження отримали значення x. Тоді розподіл імовірності для значення x для класу c обчислюється так:

$$p(X = x \mid C = c) = \frac{1}{\sqrt{2\pi s^2}} \exp\left(-\frac{\left(x - \overline{x}_c\right)^2}{2s_c^2}\right). \tag{1.22}$$

Оцінивши усі необхідні параметри для моделі, тобто апріорні розподіли імовірності належності до класу $p(C)$ та розподіли ймовірностей ознак $p(X_i \mid C)$, можна переходити до класифікації.

1.4. Побудова класифікатора з імовірнісної моделі

Наївний баєсів класифікатор поєднує баєсову імовірнісну модель з правилом прийняття рішення. Одним з поширених правил є вибір найбільш ймовірної гіпотези. Такий підхід називається *правилом прийняття рішення за максимальною апостеріорною імовірністю* (maximum a posterior (MAP) decision rule). Відповідний класифікатор є функцією classify$(X_1, \dots, X_n)$, яка визначається так:

$$\text{classify}(X_1, \dots, X_n) = \arg\max_c \left(p(C = c) \prod_{i=1}^{n} p(X_i = x_i \mid C = c) \right), \quad (1.23)$$

де $\arg\max_x f(x)$ — функція, результатом якої є множина значень x, при якому значення функції $f(x)$ максимальне (визначення 3) [3].

Визначення 3 (Функція arg max)**.** Нехай дана довільна множина X, повністю впорядкована множина Y та функція $f : X \mapsto Y$, тоді функція arg max для певного значення x над певною підмножиною S визначається так:

$$\arg\max_{x \in S \subseteq X} f(x) \coloneqq \{x \mid x \in S \wedge \forall y \in S : f(y) \leqslant f(x)\}. \quad (1.24)$$

Правило прийняття рішення за максимальною апостеріорною імовірністю правильно класифікує спостереження за умови, що ймовірність належності до правильного класу більша за ймовірності належності до інших класів, тому немає потреби у надточній оцінці цих імовірностей.

Таким чином ми отримали працюючу теоретичну модель для реалізації наївного баєсового класифікатора для класифікації спостережень, які містять неперервні (недискретні) дані.

2. Практична частина

2.1. Використані програмні засоби

Для реалізації наївного баєсового класифікатора була використана мова програмування Python версії 3.7.1. Оскільки Python — інтерпретована мова програмування, для коректної роботи розробленої реалізації необхідно встановити робочий інтерпретатор Python 3, який можна завантажити на офіційному сайті за посиланням `https://python.org/downloads`. Також розроблена реалізація використовує засоби стандартної бібліотеки мови програмування Python (табл. 1).

Табл. 1: Перелік використаних модулів стандартної бібліотеки мови програмування Python

Модуль	Призначення
`argparse`	Обробка аргументів командного рядка.
`csv`	Зчитування файлів у форматі `.csv`, які містять дані, розділені комою.
`math`	Зручне обчислення математичних функцій, зокрема e^x, a^b та $\sqrt{x}$.
`random`	Робота з випадковими числами, а саме перемішування набору даних у випадковому порядку.

2.2. Опис програми та роботи з нею

Реалізація класифікатора (яку далі ми називатимемо програмою) виконана у вигляді двох модулів на мові Python: `nbc.py` і `nbc_main.py`. Модуль `nbc.py` містить реалізації всіх функцій, необхідних для роботи наївного баєсового класифікатора, а модуль `nbc_main.py` містить інструкції, які використовують надані функції для класифікації наборів даних.

Розроблена програма виконує класифікацію неперервних чисельних даних із набору даних, який знаходиться у вхідному файлі формату CSV. Вона запускається за допомогою командного рядка таким чином:

```
1 python nbc_main.py input.csv
```

Після виконання вищезазначеної команди, програма зчитує набір даних, вказаний у файлі `input.csv`, оброблює та класифікує їх, виводить результа-

ти класифікації та її обчислену точність. Якщо вхідний файл не зазначений, програма попереджує про це користувача, надає інформацію про правильний формат використання та завершує роботу.

2.3. Опис процесу роботи програми

Модуль `nbc_main.py`, який використовується для запуску програми, містить інструкції для класифікації конкретного набору даних наївним баєсовим класифікатором. Він завантажує необхідні для роботи програми ресурси в область видимості (лістинг 2.1) та описує загальний процес роботи програми.

Лістинг 2.1: Модуль `nbc_main.py`: завантаження ресурсів, необхідних для роботи класифікатора

```
1 #!/usr/bin/env python3
2
3 import argparse
4 from nbc import *
```

Перш за все, у файлі модуля вказаний шлях до сумісного інтерпретатора, а саме Python 3. Це надає можливість запускати модуль у Unix-подібних операційних системах як скрипт, не вказуючи шлях до потрібного інтерпретатора явно.

Інструкція **`import argparse`** завантажує в область видимості модуль для обробки аргументів командного рядка. Інструкція **`from nbc import *`** завантажує всі функції, описані у модулі `nbc_main.py`, щоб їх можна було використовувати у поточному модулі.

Тепер, коли обробник параметрів командного рядка та всі функції класифікатора завантажені, відбувається перевірка способу запуску модуля (лістинг 2.2).

Лістинг 2.2: Модуль `nbc_main.py`: перевірка способу запуску модуля

```
1 if __name__ == '__main__':
2 parser = argparse.ArgumentParser(description = 'An implementation of
   ↪ Gaussian Naive Bayes Classifier in Python using stdlib facilities.
   ↪ Reads a CSV file containing floats.')
3
4 # Parse CSV dataset file
```

```
parser.add_argument('input', help = 'Path to input file')

args = parser.parse_args()

main(args)
```

Якщо модуль запущений як скрипт, наприклад, з командного рядка, як необхідно для роботи розробленої реалізації, інтерпретатор встановлює особливе значення змінної `__name__`. Тому перевіряється умова `__name__ == '__main__'`. Якщо вона виконується, ініціалізується обробник параметрів командного рядка `argparse.ArgumentParser` з коротким описом програми. Далі додається аргумент `'input'` зі стислим поясненням до нього: він містить шлях до вхідного файлу, тобто файлу з набором даних, які необхідно класифікувати.

Коли у обробнику описані бажані аргументи, створюється змінна `args`, функція `parser.parse_args()` оброблює параметри та зберігає їх значення у створеній змінній. Тепер, коли параметри оброблені, вони передаються у функцію `main()` — починаються основні етапи роботи програми (лістинг 2.3).

Лістинг 2.3: Модуль `nbc_main.py`: функція `main()`, яка описує загальний процес роботи програми

```
def main(args):
    dataset = load_training(args.input)
    dataset_train, dataset_test = split_dataset(dataset, 1/3) # split
     ↪ dataset 1 / 3

    summaries = summarize_by_class(dataset_test)
    predictions = predict_dataset(summaries, dataset_test)
    acc = compute_accuracy(dataset_test, predictions)
    print('Accuracy: {}'.format(acc))
```

Загалом, процес роботи програми можна умовно поділити на такі етапи:

1. Підготовка вхідних даних.
2. Класифікація.
3. Виведення результату.

Розглянемо кожен етап роботи програми по порядку.

2.3.1. Підготовка вхідних даних

Функція `main()` (лістинг 2.3), а з нею і етап підготовки вхідних даних, починається з інструкції `dataset = load_training(args.input)`, яка призначена для завантаження файлу з набором даних у змінну `dataset` за допомогою функції `load_training()` (лістинг 2.4). Розглянемо роботу цієї функції детальніше.

Лістинг 2.4: Модуль `nbc.py`: функція `load_training()` для завантаження набору даних у пам'ять

```
1  def load_training(filename):
2          with open(filename) as file:
3                  # ignore all lines which start with '#'
4                  reader = csv.reader(row for row in file if not
                    ↪  row.startswith('#'))
5                  dataset = []
6                  for row in reader:
7                          # comprehend each line as a list of floats and
                            ↪  append it to the dataset
8                          dataset.append([float(x) for x in row])
9
10         return dataset
```

Функція `load_training()` відкриває файл, назва якого передається в аргументі `filename`, ініціалізує CSV-зчитувач `csv.reader` у змінну `reader`. Зчитувач ініціалізується всіма рядками файлу, які не починаються з символу «#». Далі рядки, записані у зчитувач, перетворюються у вектори, які містять ознаки у форматі з плаваючою комою. Перетворені вектори записуються у змінну `dataset`. Тепер функція повертає змінну `dataset` та завершує роботу.

Далі у процесі виконання програми виконується інструкція `dataset_train, dataset_test = split_dataset(dataset, 1/3)`. Вона ділить завантажений набір даних `dataset` на тренувальний набір даних `dataset_train` і тестовий набір даних `dataset_test` за допомогою функції `split_dataset` (лістинг 2.5). Розподіл відбувається таким чином, щоб тренувальний набір даних `dataset_train` містив як мінімум `ratio` частину загального набору даних. Розбиття загального набору даних необхідне для тестування точності роботи класифікатора і є загальноприйнятою практикою в аналізі даних.

Лістинг 2.5: Модуль `nbc.py`: функція `split_dataset()` для розбиття загального набору даних на тренувальний та тестовий набори

```
1 def split_dataset(dataset, ratio):
2         dataset_shuffled = dataset
3         random.shuffle(dataset_shuffled)
4         dataset_train = dataset_shuffled[:int(len(dataset_shuffled) *
          ↪  ratio)]
5         dataset_test = dataset_shuffled[int(len(dataset_shuffled) *
          ↪  ratio):]
6
7         return dataset_train, dataset_test
```

Функція `split_dataset()` копіює отриманий набір даних `dataset` у змінну `dataset_shuffled`. Далі функція `random.shuffle()` перемішує вміст змінної `dataset_shuffled`, тобто початковий набір даних, випадковим чином, щоб усунути будь-які статистичні похибки (упередження) у даних. Перемішування відбувається безпосередньо у змінній, на місці.

Тепер дані діляться на необхідні частини: зі списку `dataset_shuffled` копіюється частина від початку до елементу з індексом `int(len(dataset_shuffled) * ratio)` і присвоюється змінній `dataset_train`. Далі з цього ж списку копіюється частина від елементу з індексом `int(len(dataset_shuffled) * ratio)` до кінця і присвоюється змінній `dataset_test` (рис. 1).

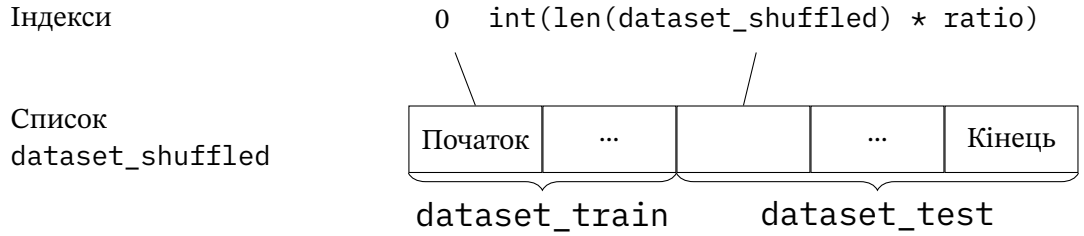


Рис. 1: Логіка розділення списку `dataset_shuffled`

Після виконання розподілу функція повертає кортеж зі змінних `dataset_train` та `dataset_test` і завершує роботу. На цьому підготовка вхідних даних завершена.

2.3.2. Класифікація

Після розбиття набору даних на тренувальний та тестовий, виконується інструкція `summaries = summarize_by_class(dataset_test)`, яка оцінює статистичні параметри значень ознак для кожного значення класової змінної за допомогою функції `summarize_by_class()` (лістинг 2.6).

Лістинг 2.6: Модуль `nbc.py`: функція `summarize_by_class()` для оцінки статистичних параметрів можливих значень кожної ознаки для усіх класів, до яких може належати спостереження

```
1  def summarize_by_class(dataset):
2          separated = separate_by_class(dataset)
3          summaries = {}
4          for class_value, data in separated.items():
5                  summaries[class_value] = summarize(data)
6
7          return summaries
```

Функція `summarize_by_class()` починає роботу з розподілу спостережень з набору даних `dataset` за класами, використовуючи функцію `separate_by_class()` (лістинг 2.7).

Лістинг 2.7: Модуль `nbc.py`: функція `separate_by_class()` для групування спостережень за класом

```
1  def separate_by_class(dataset):
2          separated = {}
3          for entry in dataset:
4                  # if entry with a given label is not yet in the
                    ↪  dictionary, add it
5                  if entry[-1] not in separated:
6                          separated[entry[-1]] = []
7
8                  separated[entry[-1]].append(entry)
9
10         return separated
```

Функція `separate_by_class()` ініціалізує пустий асоціативний масив (словник у термінології Python) `separated`. Ключем такого словника є значення класової змінної — останнє значення у даному спостереженні —

`entry[-1]`. Функція перевіряє для кожного спостереження `entry` зі списку `dataset`, чи існує в словнику `separated` ключ зі значенням `entry[-1]`. Якщо не існує, у словник додається нове значення ключа і йому у відповідність ставиться пустий список. Далі спостереження додається у словник для відповідного значення класової змінної. Функція повертає словник `separated`, що містить набір даних, в якому спостереження розподілені за класами, і завершує роботу.

Тепер управління повертається до функції `summarize_by_class()`, яка зберігає отриманий словник із згрупованими даними у словнику `separated`. Створюється пустий словник `summaries`, який міститиме статистичні характеристики вхідного набору даних для кожного класу. Далі для кожного класу `class_value` і всіх спостережень `data`, які до нього належать, зі словника `separated` виконується обчислення статистичних характеристик усіх ознак, які містяться в спостереженнях `data` за допомогою функції `summarize()` (лістинг 2.8).

Лістинг 2.8: Модуль `nbc.py`: функція `summarize()` для обчислення статистичних характеристик класу

```
1 def summarize(dataset):
2         summaries = []
3         for attribute in zip(*dataset):
4                 summaries.append((calc_mean(attribute),
                   ↪ calc_variance(attribute)))
5
6         # removes summary for last column — class
7         del summaries[-1]
8         return summaries
```

Функція `summarize()` отримує на вхід набір значень ознак `dataset`, який є списком списків. Інструкція `*dataset` «розгортає» набір значень ознак `dataset` у позиційні аргументи, а функція `zip()` створює ітератор з кортежів, в яких кожен i-й кортеж містить i-й елемент з кожної вхідної послідовності. Тобто:

$$\begin{aligned} dataset &= [[11, 12, 13], [21, 22, 23], [31, 32, 33]], \\ *dataset &= [11, 12, 13], [21, 22, 23], [31, 32, 33], \\ \text{zip}(*dataset) &= (11, 21, 31), (12, 22, 32), (13, 23, 33). \end{aligned}$$

Отже, кожне значення змінної *attribute* буде кортежем усіх значень ознак, які були у наборі спостережень.

Тепер починається оцінка параметрів, тобто обчислення статистичних характеристик набору даних: середнє значення вибірки $\overline{x}$ за допомогою функції calc_mean() (лістинг 2.9) і дисперсія вибірки s^2 за допомогою функції calc_variance() (лістинг 2.10).

Лістинг 2.9: Модуль `nbc.py`: функція `calc_mean()` для обчислення середнього значення вибірки $\overline{x}$ за (1.20)

```
1 def calc_mean(seq):
2         return sum(seq) / len(seq)
```

Лістинг 2.10: Модуль `nbc.py`: функція `calc_variance()` для обчислення дисперсії вибірки s^2 за (1.21)

```
1 def calc_variance(seq):
2         if len(seq) == 1:
3                 return 0
4         # expected value
5         mean = calc_mean(seq)
6         # the variance of the set assuming all values are equally likely
7         # use n - 1 estimator
8         variance = sum([pow(x - mean, 2) for x in seq]) / (len(seq) - 1)
9         return variance
```

Обчислені значення збираються у кортеж `(calc_mean(attribute), calc_variance(attribute))` та додаються у список `summaries`, який містить статистичні параметри кожної ознаки. Зі списку видаляється останній запис, який містить статистичні параметри класових змінних, оскільки в ньому немає потреби. Функція повертає список `summaries` і завершує роботу.

Тепер управління повертається до функції `summarize_by_class()`. По завершенню циклу **for** `class_value, data` **in** `separated.items()` словник `summaries` містить відповідності «класова змінна — статистичні характеристики ознак». Ця функція повертає словник `summaries` та передає управління функції `main()`.

У функції `main()` виконується інструкція `predictions = predict_dataset(summaries, dataset_test)`: відбувається передбачення, до яких класів належать спостереження з переданих наборів даних, за допомогою функції `predict_dataset()` (лістинг 2.11).

Лістинг 2.11: Модуль `nbc.py`: функція `predict_dataset()` для класифікації спостережень у наборі даних

```
def predict_dataset(summaries, dataset):
        predictions = []
        for entry in dataset:
                predictions.append(predict(summaries, entry))

        return predictions
```

Функція `predict_dataset()` для кожного спостереження `entry` у наборі даних `dataset` виконує класифікацію, записує результат (класову змінну) в список `predictions`, повертає його та завершує роботу. Класифікація виконується за допомогою функції `predict()` (лістинг 2.12).

Лістинг 2.12: Модуль `nbc.py`: функція `predict()` для класифікації конкретного спостереження

```
def predict(summaries, input_vector):
        probabilities = calc_class_probability(summaries, input_vector)
        # get key with max value
        return max(probabilities, key = probabilities.get)
```

Функція `predict()` виконує класифікацію конкретного спостереження `input_vector`, тобто реалізує класифікатор (1.23): вона почергово обчислює розподіли ймовірностей, що дане спостереження належить до кожного з класів, і зберігає їх у змінній `probabilities`. Далі знаходиться аргумент, при якому значення розподілу імовірності найбільше, і цей аргумент повертається.

Обчислення розподілів імовірностей виконується за допомогою функції `calc_class_probability()` (лістинг 2.13).

Лістинг 2.13: Модуль `nbc.py`: функція `calc_class_probability()` для обчислення густин імовірностей, що дане спостереження належить до кожного з класів

```
def calc_class_probability(summaries, input_vector):
        probabilities = {}
        for class_value, summaries in summaries.items():
                probabilities[class_value] = 1
```

```
5                 for (mean, variance), x in zip(summaries, input_vector):
6                         probabilities[class_value] *=
                           ↪  calculate_probability(x, mean, variance)
7
8         return probabilities
```

Функція `calc_class_probability()` реалізує модель (1.18): для кожного значення класової змінної `class_value` і її статистичних характеристик `summaries` зі словника `summaries` обчислюється ймовірність належності до первного класу і зберігається в словнику `probabilities`. Функція повертає словник `probabilities` і завершує роботу.

Обчислення ймовірності належності даного значення ознаки до певного класу виконує функція `calc_probability()` (лістинг 2.14).

Лістинг 2.14: Модуль `nbc.py`: функція `calc_probability()` для обчислення густин імовірностей, що дане спостереження належить до певного класу

```
1 def calc_probability(x, mean, variance):
2         # if variance = 0 attribute values are the same and testing
           ↪  doesn't really make sense
3         if variance == 0:
4                 return 0
5         exp = math.exp( (-math.pow(x - mean, 2) ) / (2 * variance) )
6         return (1 / math.sqrt(2 * math.pi * variance)) * exp
```

Функція `calc_probability()` обчислює густину імовірності, що дане спостереження належить до певного класу за допомогою (1.22) у два етапи, тобто спочатку обчислює значення експоненти `exp`:

$$exp = \exp\left(\frac{-\left(x-\overline{x}\right)^2}{2s^2}\right),$$

а потім і залишившуся частину виразу:

$$\frac{1}{\sqrt{2\pi s^2}} \cdot exp.$$

Тепер, після оцінки усіх параметрів моделі, та застосування наївного баєсового класифікатора, дані розподілені за класами і відповідний етап завершений.

2.3.3. Виведення результату

Коли дані класифіковані, починається аналіз і виведення результатів. Функція `main()` переходить до інструкції `acc = compute_accuracy(dataset_test, predictions)`, яка обчислює точність класифікації за допомогою функції `compute_accuracy()` (лістинг 2.15).

Лістинг 2.15: Модуль `nbc.py`: функція `compute_accuracy()` для обчислення точності класифікації

```
1 def compute_accuracy(dataset, predictions):
2         correct = 0
3         for entry, prediction in zip(dataset, predictions):
4                 print('Entry: "{}", c_act: "{}", c_pred:
                   ↪  "{}"'.format(entry, entry[-1], prediction))
5                 if entry[-1] == prediction:
6                         correct += 1
7
8         return (correct / len(dataset)) * 100.0
```

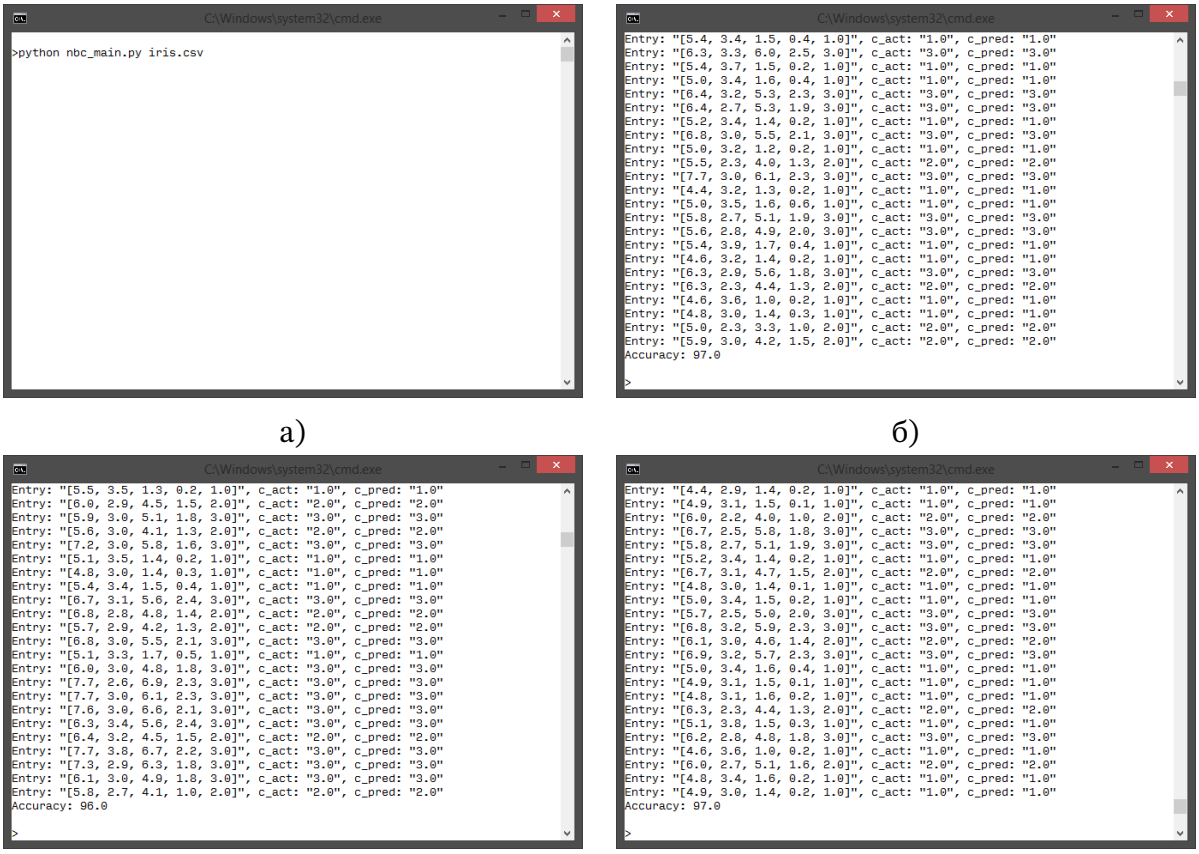
Функція `compute_accuracy()` отримує на вхід 2 аргументи: правильно класифікований тестовий набір даних `dataset`, який був виділений із загального, та список `predictions`, який містить значення класових змінних для тестового набору даних, визначених розробленим класифікатором. Для кожного спостереження виводяться наперед відомі достовірні та передбачені значення класової змінної, потім вони порівнюються і підраховуються. Перед завершенням роботи функція обчислює відсоток правильно класифікованих спостережень і повертає результат.

Після отримання значення точності класифікації, воно зберігається у змінну `acc`. Далі у функції `main()` виконується інструкція **`print`**`('Accuracy: {}'.format(acc))`, яка виводить збережене значення точності, і програма завершує роботу.

2.4. Тестування розробленої реалізації

Щоб впевнитись у правильності роботи та оцінити адекватність розробленої реалізації наївного баєсового класифікатора, необхідно її протестувати. Процес тестування полягав у запуску розробленої програми з метою класифікації набору даних, як описано в пункті 2.2, та спостереженні точності виконаної класифікації (рис. 2). Тестування проводилось у конфігурації під управлін-

ням операційної системи Microsoft Windows 8.1 з встановленим інтерпретатором Python 3.7.1, в якості тестового набору даних був використаний класичний для задач класифікації набір даних «Іриси Фішера» [5].



а) б)

в) г)

Рис. 2: Процес тестування розробленої реалізації: а — запуск програми, б–г — результати класифікації

Результат тестування показав працездатність програми і високий рівень точності на тестовому наборі даних у 3 проведених експериментах: 97,0 %, 96,0 % та 97,0 % відповідно.

Висновки

У процесі виконання даної курсової роботи була створена програмна реалізація наївного баєсового класифікатора для класифікації неперервних числових даних. Для розробки реалізації була використана мова програмування Python і засоби її стандартної бібліотеки: модулі `argparse`, `csv`, `random`. Виконуючи дану курсову роботу, я ознайомився із задачею класифікації даних, поняттям класифікатора і теоретичними відомостями про наївний баєсів класифікатор. Я дізнався про поняття моделі подій, а також ознайомився з процесом оцінки параметрів імовірнісної моделі баєсового класифікатора. Крім того, я навчився реалізовувати математичне представлення класифікатора на практиці, а також набув навичок розробки і відладки модулів, написаних на мові програмування Python.

Література

1. Statistical classification. — URL: https://en.wikipedia.org/wiki/Statistical_classification (дата зверн. 20.11.2018).
2. *Stuart Russell, Peter Norvig*. Artificial Intelligence: A Modern Approach. — 3-е вид. — Prentice Hall, 2010. — (Prentice Hall Series in Artificial Intelligence). — ISBN 9780136042594.
3. *Prof. M. Narasimha Murty, Dr. V. Susheela Devi*. Pattern Recognition: An Algorithmic Approach. — 1-е вид. — Springer-Verlag London, 2011. — (Undergraduate Topics in Computer Science 0). — ISBN 978-0-85729-494-4.
4. *Montgomery D. C.*, *Runger G. C.* Applied Statistics and Probability for Engineers. — 6-е вид. — Wiley, 2014. — ISBN 9781118539712.
5. *Fischer R. A.* Iris Data Set / за ред. M. Marshall. — 1936. — URL: https://archive.ics.uci.edu/ml/datasets/Iris (дата зверн. 20.11.2018).

Лістинг 2.16: Файл nbc_main.py

```
# This is a Naive Bayes Classifier implementation in Python
# written by Vlad Klokun and Igor Rabin
#
# Inputs: a CSV data file that contains training data
# Does: analyzes given CSV training file according to the Naive Bayes Classifier algorithm, generates random data and then makes a prediction about it
#
# Assumes a Gaussian Distribution

#!/usr/bin/env/ python3

import csv # reading CSV files
import random # list shuffling
import math # exp, pow etc

# Loads a valid CSV file containing floating-point training data into memory
# Input: a filename of a training file in CSV format
# Output: a dataset in form of a list of floats
def load_training(filename):
    with open(filename) as file:
        # ignore all lines which start with '#'
        reader = csv.reader(row for row in file if not row.startswith('#'))
        dataset = []
        for row in reader:
            # comprehend each line as a list of floats and append it to the dataset
            dataset.append([float(x) for x in row])

    return dataset

# split_dataset() splits a given dataset into 'training' and 'test' datasets to test the algorithm
# Inputs: a dataset (list), split_ratio (float)
# Outputs: a tuple of: dataset_train --- training dataset (list), dataset_test --- dataset (list)
def split_dataset(dataset, ratio):
    dataset_shuffled = dataset
```

```
34      random.shuffle(dataset_shuffled)
35      dataset_train = dataset_shuffled[:int(len(dataset_shuffled) *
        ↪ ratio)]
36      dataset_test = dataset_shuffled[int(len(dataset_shuffled) * ratio):]
37
38      return dataset_train, dataset_test
39
40  # Separate entries by class. Assumes that class is the last parameter
41  # Input: dataset
42  # Output: dictionary with class as key and matching entries as values
43  def separate_by_class(dataset):
44      separated = {}
45      for entry in dataset:
46          # if entry with a given label is not yet in the dictionary, add
            ↪ it
47          if entry[-1] not in separated:
48              separated[entry[-1]] = []
49
50          separated[entry[-1]].append(entry)
51
52      return separated
53
54  # calc_mean(): Calculates mean value of a given sequence
55  # Inputs: seq — a sequence of numbers
56  # Outputs: a mean value of the sequence (float)
57  def calc_mean(seq):
58      return sum(seq) / len(seq)
59
60  # calc_variance(): Calculates variance of a given sequence
61  # Input: seq --- a sequence of numbers
62  # Output: variance --- variance value (sigma^2) computed using n - 1
    ↪ estimator
63  def calc_variance(seq):
64      if len(seq) == 1:
65          return 0
66      # expected value
67      mean = calc_mean(seq)
68      # the variance of the set assuming all values are equally likely
69      # use n - 1 estimator
70      variance = sum([pow(x - mean, 2) for x in seq]) / (len(seq) - 1)
71      # return math.sqrt(variance)
72      return variance
73
```

```
74  # summarize(): summarizes every attribute in a dataset with their
     ↪  probabilistic properties: mean and variance. Makes no distiction
     ↪  based on class variables. returns a list of tuples which contain
     ↪  mean and variance values for respective attributes
75  # Input: dataset --- a list containing a dataset
76  # Output: summaries --- a list of tuples with statistical properties for
     ↪  each attribute in form of (mean, variance)
77  def summarize(dataset):
78      summaries = []
79      for attribute in zip(*dataset):
80          summaries.append((calc_mean(attribute),
             ↪  calc_variance(attribute)))
81
82      # removes summary for last column — class
83      del summaries[-1]
84      return summaries
85
86  # summarize_by_class(): summarizes every attribute in a dataset with
     ↪  their probabilistic properties: mean and variance. Unlike
     ↪  summarize(), the attributes are summarized on a per-class basis.
     ↪  Returns a list of tuples which contain mean and variance values for
     ↪  respective attributes belonging to certain classes.
87  # Input: dataset --- a dataset
88  # Output: summaries --- a dictionary with class variables as keys and
     ↪  statistical summaries of attributes as values
89  def summarize_by_class(dataset):
90      separated = separate_by_class(dataset)
91      summaries = {}
92      for class_value, data in separated.items():
93          summaries[class_value] = summarize(data)
94
95      return summaries
96
97  # calculate_probability(): calculates probability of a given value X
     ↪  belonging to a certain class, assuming Gaussian distribution and
     ↪  passed mean and variance values
98  # Input 1: x --- value to be tested for belonging to a certain class
99  # Input 2: mean --- mean value for an attribute which the value is
     ↪  tested against
100 # Input 2: variance --- variance value for an attribute which the value
     ↪  is tested against
101 def calculate_probability(x, mean, variance):
102     # if variance = 0 attribute values are the same and testing doesn't
         ↪  really make sense
```

```
    if variance == 0:
        return 0
    exp = math.exp( (-math.pow(x - mean, 2) ) / (2 * variance) )
    return (1 / math.sqrt(2 * math.pi * variance)) * exp

# calc_class_probability(): calculates probabilities of given input vector belonging to every class
# Input 1: summaries --- list of statistical summaries for every class
# Input 2: input_vector --- an entry to be classified
# Output: probabilities --- a dictionary containing probabilities of an entry belonging to each class
def calc_class_probability(summaries, input_vector):
    probabilities = {}
    for class_value, summaries in summaries.items():
        probabilities[class_value] = 1
        for (mean, variance), x in zip(summaries, input_vector):
            probabilities[class_value] *= calculate_probability(x, mean, variance)

    return probabilities

# predict(): predicts the class of a given entry
# Input 1: summaries --- list of statistical summaries for every class
# Input 2: input_vector --- an entry to be classified
# Output 1: key value of a class to which given entry most possibly belongs
def predict(summaries, input_vector):
    probabilities = calc_class_probability(summaries, input_vector)
    # get key with max value
    return max(probabilities, key = probabilities.get)

# predict_dataset(): makes predictions for a whole given dataset
# Input 1: summaries --- list of statistical summaries for every class
# Input 2: dataset --- a dataset to be classified
# Output 1: predictions --- a list of class values, to which entries are predicted to belong
def predict_dataset(summaries, dataset):
    predictions = []
    for entry in dataset:
        predictions.append(predict(summaries, entry))

    return predictions

```

```
141 # compute_accuracy(): computes accuracy of predictions given the dataset
    ↪ with known-good class variables
142 # Input 1: dataset --- a dataset with known-good classification
143 # Input 2: predictions --- a list of predictions made by the Native
    ↪ Bayes Classifier
144 # Output 1: accuracy in %
145 def compute_accuracy(dataset, predictions):
146     correct = 0
147     for entry, prediction in zip(dataset, predictions):
148         # c_act --- actual known-good class, c_pred --- class predicted
            ↪ by Naive Bayes Classifier
149         print('Entry: "{}", c_act: "{}", c_pred: "{}"'.format(entry,
            ↪ entry[-1], prediction))
150         if entry[-1] == prediction:
151             correct += 1
152
153     return (correct / len(dataset)) * 100.0
```

Лістинг 2.17: Файл nbc.py

```
1  #!/usr/bin/env python3
2
3  import argparse
4  from nbc import *
5
6  def main(args):
7      dataset = load_training(args.input)
8      dataset_train, dataset_test = split_dataset(dataset, 1/3) # split
       ↪ dataset 1 / 3
9
10     summaries = summarize_by_class(dataset_test)
11     predictions = predict_dataset(summaries, dataset_test)
12     acc = compute_accuracy(dataset_test, predictions)
13     print('Accuracy: {}'.format(acc))
14
15 if __name__ == '__main__':
16     parser = argparse.ArgumentParser(description = 'An implementation of
       ↪ Gaussian Naive Bayes Classifier in Python using stdlib
       ↪ facilities. Reads a CSV file containing floats.')
17
18     # Parse CSV dataset file
19     parser.add_argument('input', help = 'Path to input file')
20
```

```
21      args = parser.parse_args()
22
23      main(args)
```

Додаток Б. Набір даних «Іриси Фішера»

Лістинг 2.18: Файл `iris.csv`

```
5.1,3.5,1.4,0.2,1
4.9,3.0,1.4,0.2,1
4.7,3.2,1.3,0.2,1
4.6,3.1,1.5,0.2,1
5.0,3.6,1.4,0.2,1
5.4,3.9,1.7,0.4,1
4.6,3.4,1.4,0.3,1
5.0,3.4,1.5,0.2,1
4.4,2.9,1.4,0.2,1
4.9,3.1,1.5,0.1,1
5.4,3.7,1.5,0.2,1
4.8,3.4,1.6,0.2,1
4.8,3.0,1.4,0.1,1
4.3,3.0,1.1,0.1,1
5.8,4.0,1.2,0.2,1
5.7,4.4,1.5,0.4,1
5.4,3.9,1.3,0.4,1
5.1,3.5,1.4,0.3,1
5.7,3.8,1.7,0.3,1
5.1,3.8,1.5,0.3,1
5.4,3.4,1.7,0.2,1
5.1,3.7,1.5,0.4,1
4.6,3.6,1.0,0.2,1
5.1,3.3,1.7,0.5,1
4.8,3.4,1.9,0.2,1
5.0,3.0,1.6,0.2,1
5.0,3.4,1.6,0.4,1
5.2,3.5,1.5,0.2,1
5.2,3.4,1.4,0.2,1
4.7,3.2,1.6,0.2,1
4.8,3.1,1.6,0.2,1
5.4,3.4,1.5,0.4,1
5.2,4.1,1.5,0.1,1
5.5,4.2,1.4,0.2,1
4.9,3.1,1.5,0.1,1
5.0,3.2,1.2,0.2,1
5.5,3.5,1.3,0.2,1
4.9,3.1,1.5,0.1,1
4.4,3.0,1.3,0.2,1
5.1,3.4,1.5,0.2,1
```

```
5.0,3.5,1.3,0.3,1
4.5,2.3,1.3,0.3,1
4.4,3.2,1.3,0.2,1
5.0,3.5,1.6,0.6,1
5.1,3.8,1.9,0.4,1
4.8,3.0,1.4,0.3,1
5.1,3.8,1.6,0.2,1
4.6,3.2,1.4,0.2,1
5.3,3.7,1.5,0.2,1
5.0,3.3,1.4,0.2,1
7.0,3.2,4.7,1.4,2
6.4,3.2,4.5,1.5,2
6.9,3.1,4.9,1.5,2
5.5,2.3,4.0,1.3,2
6.5,2.8,4.6,1.5,2
5.7,2.8,4.5,1.3,2
6.3,3.3,4.7,1.6,2
4.9,2.4,3.3,1.0,2
6.6,2.9,4.6,1.3,2
5.2,2.7,3.9,1.4,2
5.0,2.0,3.5,1.0,2
5.9,3.0,4.2,1.5,2
6.0,2.2,4.0,1.0,2
6.1,2.9,4.7,1.4,2
5.6,2.9,3.6,1.3,2
6.7,3.1,4.4,1.4,2
5.6,3.0,4.5,1.5,2
5.8,2.7,4.1,1.0,2
6.2,2.2,4.5,1.5,2
5.6,2.5,3.9,1.1,2
5.9,3.2,4.8,1.8,2
6.1,2.8,4.0,1.3,2
6.3,2.5,4.9,1.5,2
6.1,2.8,4.7,1.2,2
6.4,2.9,4.3,1.3,2
6.6,3.0,4.4,1.4,2
6.8,2.8,4.8,1.4,2
6.7,3.0,5.0,1.7,2
6.0,2.9,4.5,1.5,2
5.7,2.6,3.5,1.0,2
5.5,2.4,3.8,1.1,2
5.5,2.4,3.7,1.0,2
5.8,2.7,3.9,1.2,2
6.0,2.7,5.1,1.6,2
```

```
5.4,3.0,4.5,1.5,2
6.0,3.4,4.5,1.6,2
6.7,3.1,4.7,1.5,2
6.3,2.3,4.4,1.3,2
5.6,3.0,4.1,1.3,2
5.5,2.5,4.0,1.3,2
5.5,2.6,4.4,1.2,2
6.1,3.0,4.6,1.4,2
5.8,2.6,4.0,1.2,2
5.0,2.3,3.3,1.0,2
5.6,2.7,4.2,1.3,2
5.7,3.0,4.2,1.2,2
5.7,2.9,4.2,1.3,2
6.2,2.9,4.3,1.3,2
5.1,2.5,3.0,1.1,2
5.7,2.8,4.1,1.3,2
6.3,3.3,6.0,2.5,3
5.8,2.7,5.1,1.9,3
7.1,3.0,5.9,2.1,3
6.3,2.9,5.6,1.8,3
6.5,3.0,5.8,2.2,3
7.6,3.0,6.6,2.1,3
4.9,2.5,4.5,1.7,3
7.3,2.9,6.3,1.8,3
6.7,2.5,5.8,1.8,3
7.2,3.6,6.1,2.5,3
6.5,3.2,5.1,2.0,3
6.4,2.7,5.3,1.9,3
6.8,3.0,5.5,2.1,3
5.7,2.5,5.0,2.0,3
5.8,2.8,5.1,2.4,3
6.4,3.2,5.3,2.3,3
6.5,3.0,5.5,1.8,3
7.7,3.8,6.7,2.2,3
7.7,2.6,6.9,2.3,3
6.0,2.2,5.0,1.5,3
6.9,3.2,5.7,2.3,3
5.6,2.8,4.9,2.0,3
7.7,2.8,6.7,2.0,3
6.3,2.7,4.9,1.8,3
6.7,3.3,5.7,2.1,3
7.2,3.2,6.0,1.8,3
6.2,2.8,4.8,1.8,3
6.1,3.0,4.9,1.8,3
```

```
6.4,2.8,5.6,2.1,3
7.2,3.0,5.8,1.6,3
7.4,2.8,6.1,1.9,3
7.9,3.8,6.4,2.0,3
6.4,2.8,5.6,2.2,3
6.3,2.8,5.1,1.5,3
6.1,2.6,5.6,1.4,3
7.7,3.0,6.1,2.3,3
6.3,3.4,5.6,2.4,3
6.4,3.1,5.5,1.8,3
6.0,3.0,4.8,1.8,3
6.9,3.1,5.4,2.1,3
6.7,3.1,5.6,2.4,3
6.9,3.1,5.1,2.3,3
5.8,2.7,5.1,1.9,3
6.8,3.2,5.9,2.3,3
6.7,3.3,5.7,2.5,3
6.7,3.0,5.2,2.3,3
6.3,2.5,5.0,1.9,3
6.5,3.0,5.2,2.0,3
6.2,3.4,5.4,2.3,3
5.9,3.0,5.1,1.8,3
```