

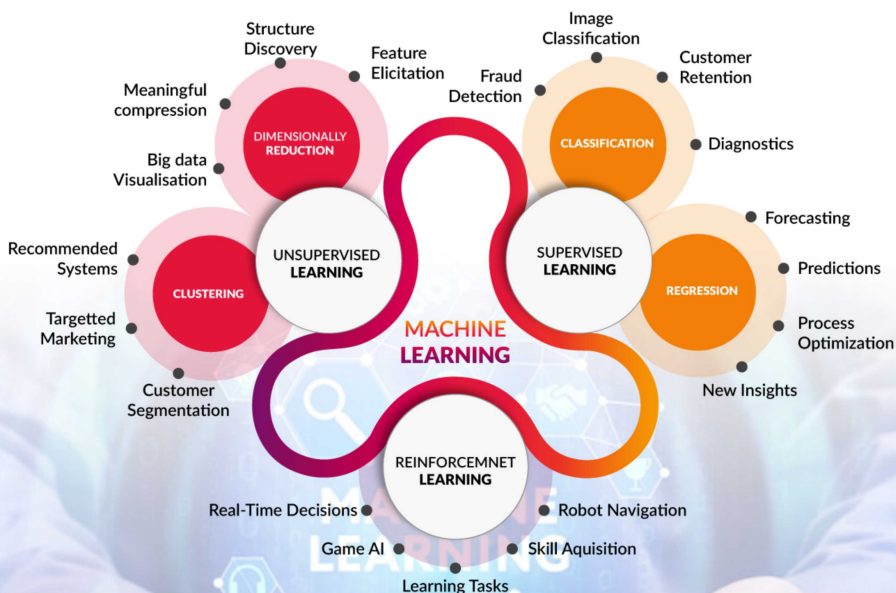


НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

К.С. Зайцев, М.Е. Дунаев

Использование методов машинного обучения и языка Python для анализа данных

Часть 1



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

К.С. Зайцев, М.Е. Дунаев

**Использование методов машинного обучения
и языка *Python* для анализа данных**

Часть 1

Лабораторный практикум

Москва, 2019

УДК [004.6:004.9](076)
ББК 32.973.26-018.2я7
3-17

Зайцев К.С., Дунаев М.Е. **Использование методов машинного обучения и языка *Python* для анализа данных. Часть 1:** Лабораторный практикум. – М.: НИЯУ МИФИ, 2019. – 48 с.

Лабораторный практикум ориентирован на бакалавров и магистров, специализирующихся по направлениям «Информатика и вычислительная техника», «Программная инженерия», «Информационная безопасность» и «Бизнес-информатика», в части практического применения современных методов машинного обучения для анализа данных в прикладных системах, управляемых данными, различного назначения.

В части 1 практикума изучаемые методы машинного обучения разбиты на три группы: методы обучения без учителя (*Unsupervised Learning*); методы обучения с учителем (*Supervised Learning*) и методы рекомендательных систем (*Recommended Systems*). По каждому изучаемому методу в практикуме приведена *Python*-программа, использующая популярные библиотеки, и несколько заданий для самостоятельной проработки.

Среди исследуемых методов в этой части практикума рассматриваются:
метод главных компонент (*Principal Component Analysis, PCA*);
методы иерархической (агломеративный) и неиерархической (*k*-средних) кластеризации;
методы деревьев решений; ближайших соседей и линейной регрессии;
алгоритмы поиска ассоциативных правил *Apriori* и *FP-growth*;
алгоритмы коллаборативной фильтрации (сингулярного разложения матриц (*Singular Value Decomposition, SVD*)).

Рецензент доцент А.А. Дюмин

ISBN 978-5-7262-2561-6

© Национальный исследовательский
ядерный университет «МИФИ», 2019
© К.С. Зайцев, 2019

Редактор Н.Н. Антонова

Подписано в печать 23.04.2019. Формат 60х84 1/16

П.л. 3,0. Уч.-изд.л. 3,0.

Изд. № 023-1. Заказ № 45.

Национальный исследовательский ядерный университет «МИФИ».

Типография НИЯУ МИФИ.

115409, Москва, Каширское шоссе, 31

ОГЛАВЛЕНИЕ

Введение	4
1. Лабораторная работа № 1. Программные инструменты, необходимые для выполнения лабораторных работ по машинному обучению на языке Python	7
1.1. Библиотеки и версии	7
1.2. Установка программных инструментов и некоторые операции	7
1.3. Домашнее задание	13
1.4. Материалы для изучения Python	14
2. Лабораторная работа № 2. Машинное обучение без учителя. Методы снижения размерности и кластеризации	14
2.1. Машинное обучение без учителя	14
2.2. Снижение размерности пространства данных. Метод главных компонент (<i>Principal Component Analysis, PCA</i>)	15
2.3. Кластеризация	21
2.4. Метрики качества кластеризации	25
2.5. Домашнее задание	26
2.6. Контрольные вопросы	27
3. Лабораторная работа № 3. Машинное обучение с учителем. Методы классификации (<i>Classification</i>) и регрессии (<i>Regression</i>)	27
3.1. Машинное обучение с учителем	27
3.2. Методы классификации	28
3.3. Методы регрессии	30
3.4. Домашние задания	32
3.5. Контрольные вопросы	34
4. Лабораторная работа № 4. Рекомендательные системы. Методы построения ассоциативных правил и коллаборативной фильтрации	35
4.1. Персонализация онлайн-маркетинга	35
4.2. Поиск ассоциативных правил	36
4.3. Методы коллаборативной фильтрации	40
4.4. Домашние задания	44
4.5. Контрольные вопросы	46
Список литературы	47

Введение

В последнее время, анализируя успехи в бизнесе организаций, широко использующих в своей деятельности компьютерные технологии обработки больших данных (*bigdata*), начали говорить об организациях, управляемых данными (*data driven*). Успехи таких организаций связывают с широким использованием методов машинного обучения (*machine learning*, *ML*) и искусственного интеллекта (*artificial intelligence*, *AI*), преобразующих сырые (*raw*) данные, полученные из различных источников в ценный ресурс для принятия решений (рис. в.1). И, чем больше мы преобразуем данные, тем они становятся более дорогими для нас.



Рис. в.1. Схема преобразования данных в ценный ресурс управления.

Что означает быть организацией, управляемой данными? Многие руководители считают, что, поскольку их компания создает множество отчетов, и использует панели индикаторов (*dashboard*), то она как раз и является управляемой данными [1]. Однако и отчеты, и панели индикаторов по своей сути ретроспективны. Они могут сообщить нам, что случилось на прошлой неделе, или какие индикаторы это подтверждают, но не скажут: почему это произошло и что с этим делать дальше?

Ни статистические отчеты, ни панели индикаторов не делают организацию управляемой данными, так как такая организация обязана формировать аналитические заключения или аналитики, нацеленные на будущее. Компании нужны ответы на вопросы: «Почему это произошло?», «Кто не справился с заданием?» и «Что делать дальше?» с соответствующими обоснованиями и сценариями «что, если» дальнейших действий.

Отличительным признаком организации управляемой данными является эффективная цепочка создания стоимости (*value chain*) с помощью аналитических заключений. Эта цепочка представляет собой циклический процесс: применение аналитики приводит к изменениям в бизнесе, результаты изменений оцениваются, и эта информация подается на вход следующей итерации анализа.

Под аналитикой здесь понимаются прогнозирующие и объясняющие ход рассуждения модели, полученные с применением методов машинного обучения, включая глубокое обучение (*deep learning, DL*) и логического вывода к внутренним и внешним базам данных компании, для поддержки принимаемых решений в операционных процессах.

В зависимости от цели аналитику принято разделять на описательную (дескриптивную), предсказательную (предикативную) и предписывающую (прескриптивную) [2].

Для получения *описательной* аналитики используют только внутренние данные компании. Она включает все виды отчетности и панели индикаторов, которые применяются при управлении компанией. При получении результатов данного вида аналитики методы машинного обучения практически не используются.

Главным отличием двух других видов аналитики от описательной является то, что они используют все имеющиеся в компании внутренние и внешние данные, работая фактически с *bigdata*.

Целью *предсказательной* аналитики являются выявление причин отдельных событий в деятельности коллективов людей или устройств, путем извлечения неочевидных зависимостей в данных. Для построения сценариев и моделей функционирования компании на этом этапе широко используются методы машинного обучения.

Предписывающая аналитика позволяет не только выявлять причины, приводящие в перспективе к определенным событиям в деятельности компании, но и определять последовательности действий, способных их предотвратить. Этот тип аналитики позволяет принимать решения в реальном времени. Для ее реализации используются

методы машинного обучения и другие методы искусственного интеллекта.

Если мы рассмотрим топ-10 трендов в сфере анализа данных, представленных компанией *Gartner*, то увидим, что основное внимание в ближайшие годы будет направлено на расширенную или дополненную аналитику (*augmented analytics*) и объединение всех собираемых для анализа данных в фабрики (*Data Factory*) или озера (*Data Lake*) данных [3]. Предполагается, что именно в этом секторе информационных технологий в ближайшее время будет наибольший спрос на кадры, владеющие методами машинного обучения, и способными применять их для новых систем бизнес-анализа, платформ исследования данных и машинного обучения, а также продуктов со встроенными аналитическими системами.

Расширенная аналитика включает три различных области или этапа развития [4]:

1) расширенная подготовка данных, т.е. использование алгоритмов искусственного интеллекта, включая машинное обучение для сбора, организации и интеграции данных, поступающих из разных источников;

2) диалоговая (*conversational*) расширенная аналитика – использование неспециалистами в области анализа данных методов машинного обучения для автоматического поиска и визуализации наиболее релевантных результатов без построения аналитических моделей;

3) расширенная наука о данных - технологии искусственного интеллекта, включая машинное обучение, смогут автоматически генерировать и управлять моделями расширенного анализа.

Как видно процесс развития и совершенствования человеческой деятельности во всех областях будет связан с анализом все увеличивающихся объемов данных для эффективного принятия решений. Поэтому необходимо представлять себе границы применимости и области рационального использования различных групп методов машинного обучения.

В настоящем лабораторном практикуме изучаются широко распространенные методы машинного обучения без учителя (*Unsupervised Learning*), с учителем (*Supervised Learning*) и методы создания рекомендательных систем (*Recommended Systems*).

1. Лабораторная работа № 1. Программные инструменты, необходимые для выполнения лабораторных работ по машинному обучению на языке *Python*

Цель работы: установка и практическое изучение программных инструментов, необходимых для дальнейшего выполнения комплекса лабораторных работ по курсу «Машинное и глубокое обучение».

1.1 Библиотеки и версии

При использовании методов машинного обучения для решения различных практических задач, как правило, используют язык *Python* в интерактивной среде разработки (*Interactive Development Environment, IDE*), позволяющей быстро корректировать имеющуюся программу, и набор библиотек для расчетов и визуализации результатов. Для выполнения лабораторных работ нам понадобятся следующие инструменты:

- *Python* (версии 2.7 или 3.5, в зависимости от личных предпочтений);

- библиотеки языка *Python*:

NumPy – для работы с многомерными массивами данных, включая матрицы,

SciPy – для вычисления математических функций (интерполяция, интегрирование, дифференцирование, обработка сигналов и т.д.),

Pandas - для манипулирования числовыми таблицами и временными рядами,

Matplotlib – для визуализация данных,

Scikit-Learn – для использования методов машинного обучения (сокращения размерности, кластеризации, регрессии и т.д.).

Эти библиотеки можно установить с помощью *pip* - менеджера пакетов, написанных на *Python*, из терминала:

`pip install numpy scipy pandas scikit-learn;`


- интерактивная среда разработки *Anaconda*. Можно также использовать среду *PyCharm*, но здесь рекомендуется *Anaconda*..

1.2 Установка программных инструментов и некоторые операции

Установка среды *Anaconda*

Будем работать в одной из операционных систем: *MS Windows*, *Linux* или *macOS*. Для этого перейдем по ссылке: <https://www.anaconda.com/distribution/#download-section>. На откры-

вающихся последовательно экранов сценария установки выполним следующие действия.

- 1) Нажмем, например, экранную кнопку  **Windows** для работы в ОС *MS Windows*, после чего выберем нужную версию *Python* в соответствии с разрядностью ОС (рис. 1.1).

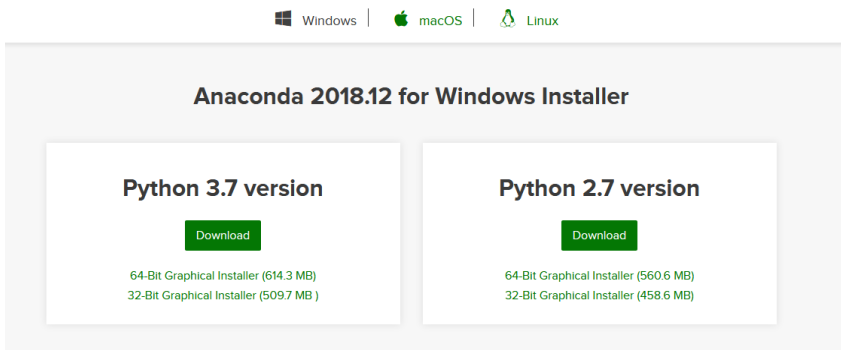


Рис. 1.1. Выбор версии языка Python.

- 2) Скачаем установщик, и после завершения загрузки запустим его от имени администратора.
- 3) Последовательно нажмем кнопки *Next->I Agree*.
- 4) Выберем *Just Me* и нажмем *Next* (рис. 1.2).

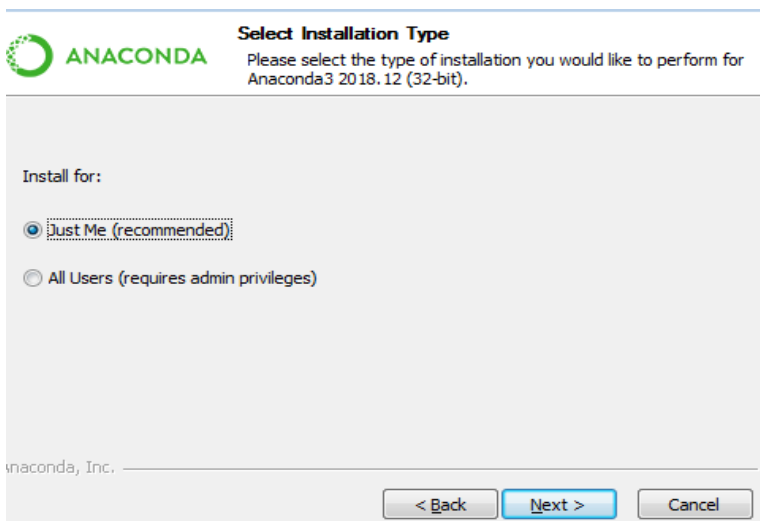


Рис. 1.2. Выбор типа пользователя.

- 5) Укажем директорию для сохранения и нажмем *Next* (рис. 1.3).

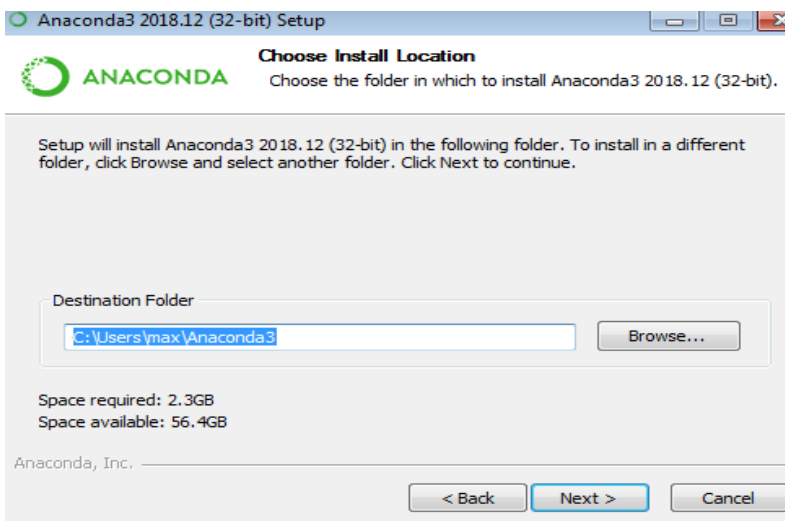


Рис. 1.3. Выбор директории для сохранения.

- 6) Установим обе галочки в полномочиях, и нажмем **Install** (рис. 1.4).

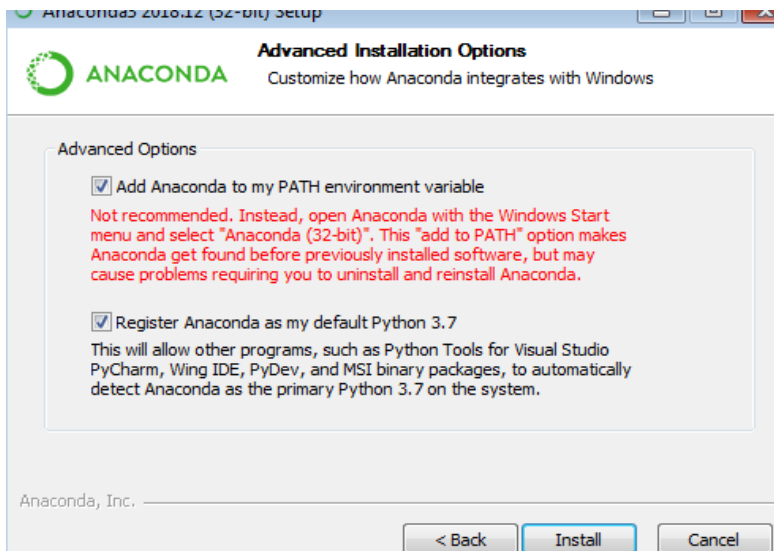


Рис. 1.4. Инсталляция.

- 7) Нажмем **Next** (рис. 1.5)

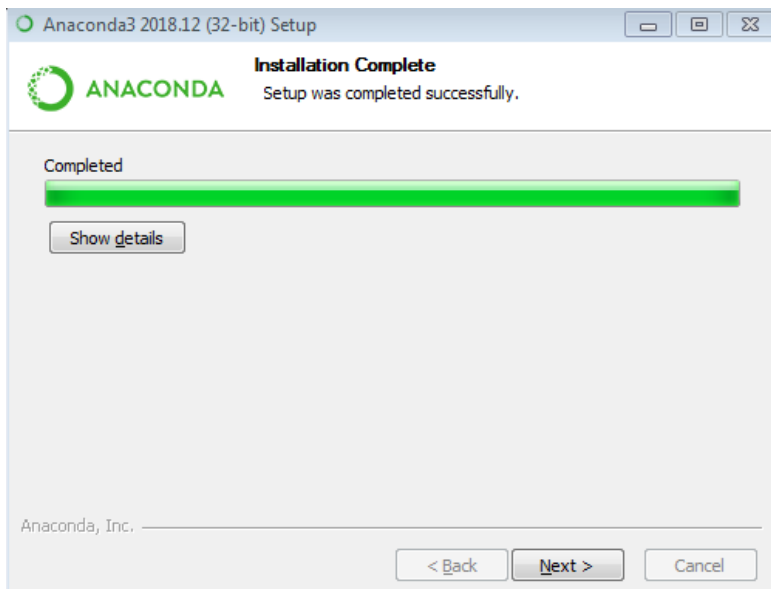


Рис. 1.5. Завершение инсталляции.

8) Нажмем *Skip* (рис. 1.6)

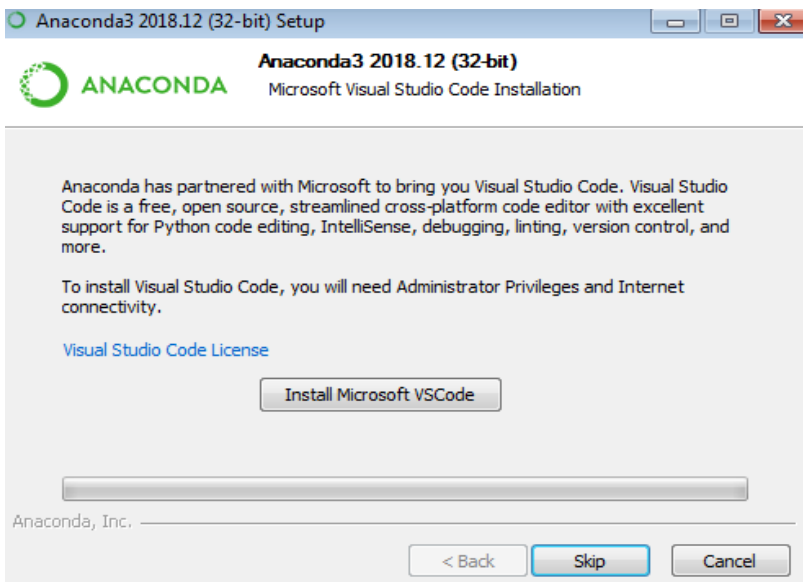


Рис. 1.6. Anaconda3 установлена.

9) Нажмем *Finish*.

После завершения в установленной папке найдем файл *Spyder* – это и есть наша среда разработки (рис. 1.7).

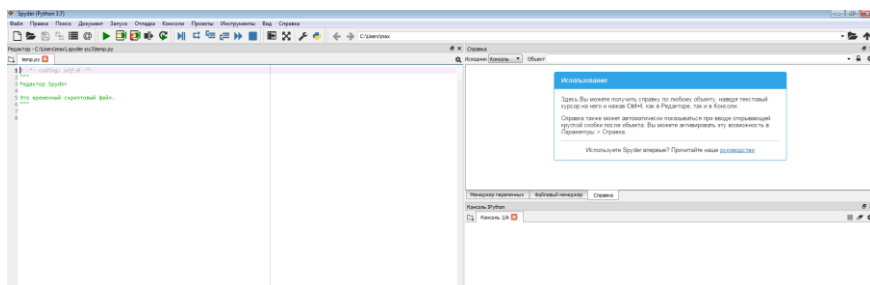


Рис. 1.7. Среда разработки Spyder.

Установка библиотеки NumPy

Для импорта библиотеки *NumPy* выполним команду: `import numpy`. Чтобы каждый раз не писать в тексте программы слово “*numpy*” можно заменить его любым сокращением, например, *np*: `import numpy as np`.

Генерация случайной матрицы

Сгенерируем матрицу, состоящую из 100 строк и 10 столбцов, элементы которой являются случайной (*random*) выборкой из нормального распределения $N(10, 100)$.

Функция для генерации чисел из нормального распределения: *np.random.normal*. Ее параметры:

- *loc*: среднее нормального распределения (в нашем случае 10),
- *scale*: стандартное отклонение нормального распределения (в нашем случае 10),
- *size*: размер матрицы (в нашем случае (100, 10)).

Код для самопроверки

```
X = np.random.normal(loc=10, scale=10, size=(100, 10))
print(X)
```

Нормировка матрицы

Произведем нормировку матрицы: вычитая из элементов среднее, и деля на стандартное отклонение. Функция для вычисления среднего: *np.mean*. Функция для вычисления стандартного отклонения: *np.std*. Первый параметр этих функций — матрица, для которой производятся вычисления.

Также полезным будет параметр *axis*, который указывает, по какому измерению вычисляются среднее и стандартное отклонение

(если *axis*=0, то по столбцам, если *axis*=1, то по строкам; если его не указывать, то данные величины будут вычислены по всей матрице).

Код для самопроверки

```
mean = np.mean(X, axis=0)
sko = np.std(X, axis=0)
normX = ((X - mean) / sko)
print(normX)
```

Операции над элементами матрицы

Выведем для заданной матрицы номера строк, сумма элементов в которых превосходит 10.

Функция для подсчета суммы: ***np.sum***

Аргументы аналогичны функциям ***np.mean*** и ***np.std***.

К матрицам можно применять логические операции, которые будут применяться поэлементно. Соответственно, результатом такой операции будет матрица такого же размера, в ячейках которой будет записано либо *True*, либо *False*. Индексы элементов со значением *True* можно получить с помощью функции ***np.nonzero***.

Заданная матрица

```
Z = np.array([[4, 5, 0],
               [1, 9, 3],
               [5, 1, 1],
               [3, 3, 3],
               [9, 9, 9],
               [4, 7, 1]])
```

Код для самопроверки

```
r = np.sum(Z, axis=1)
print(np.nonzero(r > 10))
```

Объединение матриц

Сгенерируем две единичные матрицы (с единицами на главной диагонали) размера 3x3. Соединим две матрицы в одну размера 6x3.

Функция для генерации единичной матрицы: ***np.eye***

Аргумент: число строк (или, что эквивалентно, столбцов).

Функция для вертикальной стыковки матриц: ***np.vstack((A, B))***

Код для самопроверки

```
A = np.eye(3)
B = np.eye(3)
print(A)
```

```
print(B)
AB = np.vstack((A, B))
print(AB)
```

1.3 Домашнее задание

Скажем несколько слов о структурах хранения данных в *Pandas*, которую будем использовать в этом домашнем задании.

Основными являются *Series* и *DataFrame*. *Series* – это проиндексированный одномерный массив значений. Он похож на простой словарь типа *dict*, где имя элемента будет соответствовать индексу, а значение – значению записи. *DataFrame* — это проиндексированный многомерный массив значений, соответственно каждый столбец *DataFrame*, является структурой *Series*.

Прежде, чем работать с данными, научимся их загружать. Для этого воспользуемся методом *read_csv* из библиотеки *Pandas*:

```
import pandas
data = pan-
das.read_csv(filepath_or_buffer='D:/Dataset/titanic.csv')
```

Данные будут представлены в виде *DataFrame*. Чтобы выбрать один из столбцов фрейма данных следует написать название столбца в квадратных скобках, например, для печати

```
print(data['Pclass']) .
```

Все ранее рассмотренные методы, такие как получение среднего или стандартного отклонения и т.д., применимы так же и к *DataFrame*, например, *data.mean* .

Инструкция по выполнению

Изучив данные о пассажирах Титаника из датасета *titanic.csv* (ссылка: <https://github.com/tpo9hbi4/MachineDeepLearning>) и используя описанные выше методы, необходимо дать ответы на вопросы:

1. Сколько мужчин было на корабле?
2. Какая доля пассажиров (в %) выжила?
3. Какая доля пассажиров (в %) от общего количества путешествовала во 2-ом классе?
4. Посчитайте среднее и медиану возраста всех людей на корабле.
5. Коррелируют ли число братьев/сестер с числом родителей/детей? Посчитайте корреляцию по Пирсону между признаками *SibSp* и *Parch* датасета.

6. Какое самое популярное женское имя было на корабле? Извлеките из полного имени пассажира (колонок *Name*) его личное имя (*FirstName*).

1.4 Материалы для изучения *Python*

Рекомендуемая книга: [Python for Data Analysis](#) (перевод [Python и анализ данных](#)) от автора библиотеки *Pandas*. В книге приведена вся необходимая информация для выполнения заданий курса.

Python-пакеты для курса

- [NumPy](#): работа с массивами и матрицами,
- [Pandas](#): манипулирование таблицами и временными рядами,
- [Matplotlib](#): визуализация данных,
- [SciKit-Learn](#): библиотека алгоритмов машинного обучения,
- [SciPy](#): вспомогательные математические функции.

Полезные ссылки

- [Официальный сайт Python](#)
- [Коллекция интересных IPython-ноутбуков \(блокнотов\)](#)
- Лекции по научным вычислениям с Python [Scientific Python](#)
- Ускоренный курс по *Python* для ученых [A Crash Course in Python for Scientists](#)
- 100+ бесплатных книг по науке о данных [100+ Free Data Science Books](#)

2. Лабораторная работа № 2. Машинное обучение без учителя. Методы снижения размерности (*Dimensionally Reduction*) и кластеризации (*Clustering*)

Цель работы: изучение на практике метода главных компонент (*Principal Component Analysis* или *PCA*) для снижения размерности признакового пространства, а также иерархического (агломеративного) и неиерархического (*KMeans*) методов кластеризации, их особенностей и возможностей эффективного использования. Эти методы относятся к направлению машинного обучения без учителя.

2.1 Машинное обучение без учителя

Основное отличие методов машинного обучения без учителя в том, что в этом случае нет разметки экспертом обрабатываемых данных. Поэтому проявляется несколько особенностей использования этих методов:

- возможность использовать большие объёмы данных для настройки и проверки алгоритмов, т.к. их не нужно размечать руками для обучения;

- отсутствие ясности при измерении качества применяемых методов, из-за отсутствия интуитивно понятных метрик, используемых в методах обучения с учителем.

Основными задачами, решаемыми этими методами, являются: сокращение размерности пространства данных, построение кластеров и поиска ассоциаций.

Все массивы данных для обработки и тексты программ, реализующих методы снижения размерности и кластеризации, для данной лабораторной работы можно скачать по ссылке

<https://github.com/tpo9hbi4/MachineDeepLearning/tree/master/PCAandClustering>

2.2 Снижение размерности пространства данных. Метод главных компонент (*Principal Component Analysis, PCA*)

Метод главных компонент [5] является наиболее популярным методом снижения размерности пространства, в котором заданы множества данных (часто говорят датасеты). Сначала мы применим этот метод для перехода из трехмерного пространства ($3D$) в двухмерное ($2D$) и визуализации этих пространств с данными.

Для этого вслед за [6] рассмотрим классический пример про цветы ирисы, написанный на языке *Python*, и на датасете описаний цветов произведем снижение размерности пространства признаков.

Загрузим все модули, необходимые для проведения работы, а именно: пакет *sklearn* для загрузки датасетов и методов машинного обучения, пакеты *mpl_toolkits.mplot3d*, *matplotlib* для работы с $3D$ и $2D$ графикой, а так же библиотеку *numpy* для работы с матрицами.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns;
sns.set(style='white')
from sklearn import datasets
from mpl_toolkits.mplot3d import Axes3D
```

```
#Далее загрузим датасет с описательными признаками
цветов ирисов
```

```
iris = datasets.load_iris()
```

```
#Выделим матрицу данных цветов (4 штуки) и столбец от-
ветов, содержащий 3 класса ирисов: 'Setosa' с кодом
```

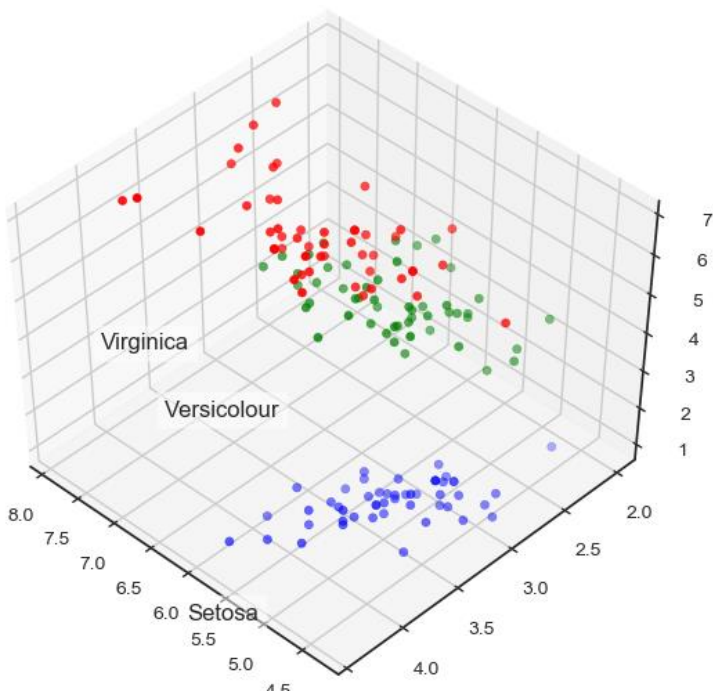



Рис 2.2. Изображение данных размерности 3D

Теперь снизим размерность этого пространства до 2D, используя метод PCA:

```
#Загрузим пакет с методом PCA
from sklearn import decomposition
pca = decomposition.PCA(n_components=2)
#Выполним центровку данных, путем вычитания среднего по
#столбцам
X_centered = X - X.mean(axis=0)
X_pca = pca.fit_transform(X_centered)
#размерность изменилась до 2D
#Нарисуем получившиеся точки в пространстве 2D см. рис.
#2.3.
plt.plot(X_pca[y == 0, 0], X_pca[y == 0, 1], 'bo', la-
bel='Setosa')
plt.plot(X_pca[y == 1, 0], X_pca[y == 1, 1], 'go', la-
bel='Versicolour')
plt.plot(X_pca[y == 2, 0], X_pca[y == 2, 1], 'ro', la-
bel='Virginica')
```

```
plt.l+nd(loc=0);
plt.show()
```

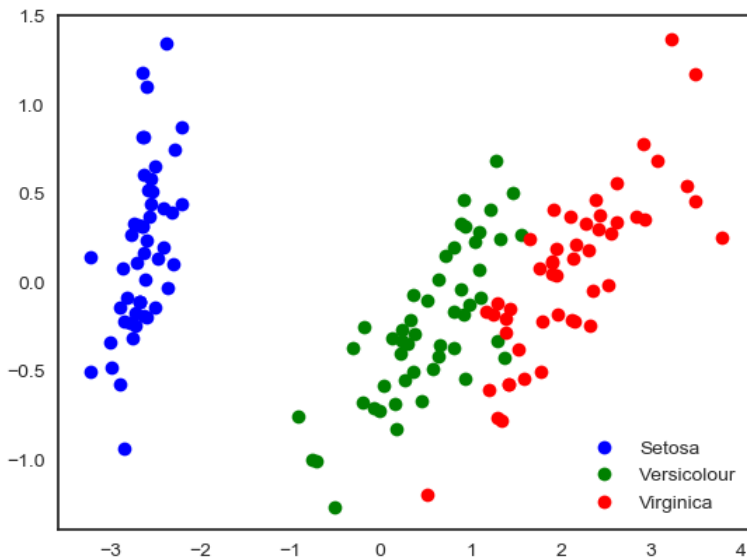


Рис. 2.3. Отображение данных, после снижения размерности до 2D.

Видно, что картинки в 2D и в 3D имеют схожую структуру. Иногда снижение размерности даже улучшает последующую работу алгоритмов машинного обучения. Рассмотренный пример позволяет создать дерево решений для задачи классификации по 2D выборке. И в этом случае точность по сравнению с 3D выборкой даже несколько увеличится с 0.88889 до 0.91111.

Теперь возьмем другой набор данных с изображением рукописных цифр (`digits`), где каждый элемент представлен строкой из 64 значений цветов с кодами от 0 (черный) до 16 (белый).

```
digits = datasets.load_digits()
X = digits.data
y = digits.target
#Выведем матрицу с объектами и признаками рис.2.4.
#Свернём признаковое описание в матрицу цветов 8x8
и изобразим эти рукописные цифры см. рис. 2.5
plt.figure(figsize=(16, 6))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X[i,:].reshape([8,8]));
```

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
[0 1 2 ... 8 9 8]
```

Рис 2.4. Данные датасета с цифрами.

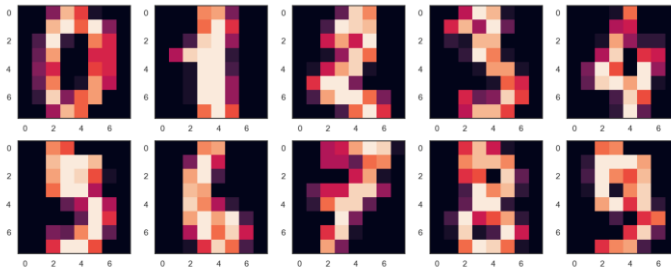


Рис.2.5. Отображение данных с цифрами от 0 до 9.

```
#Уменьшим размерность пространства с 64 координат до 2
pca = decomposition.PCA(n_components=2)
X_reduced = pca.fit_transform(X)
#Отообразим элементы датасета в новом 2-х мерном про-
странстве, раскрасив их в соответствии со значением y
(см. рис. 2.6)
plt.figure(figsize=(12,10))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y,
            edgecolor='none', alpha=0.7, s=40,
            cmap=plt.cm.get_cmap('nipy_spectral', 10))
plt.colorbar()
```

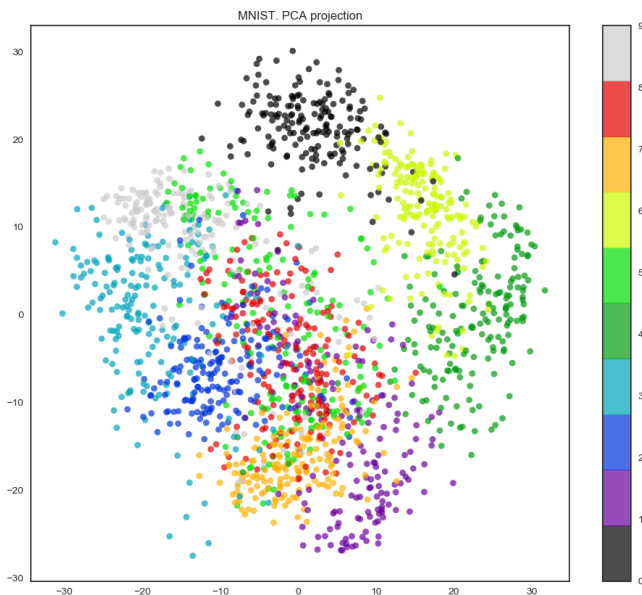


Рис 2.6. Отображение данных сниженной размерности.

На практике, как правило, выбирают столько главных компонент, чтобы оставить 90% дисперсии исходных данных (см. рис.2.7). В данном случае для этого достаточно выделить 21 главный компонент, то есть снизить размерность с 64 признаков до 21.

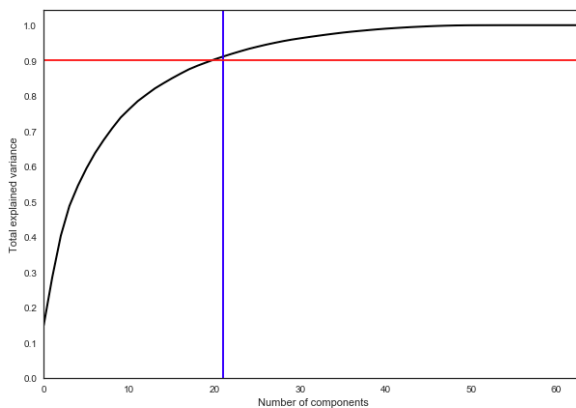


Рис 2.7. Выбор количества компонент в методе PCA.

```
pca = decomposition.PCA().fit(X)
plt.figure(figsize=(10,7))
```

```
plt.plot(np.cumsum(pca.explained_variance_ratio_),
color='k', lw=2)
plt.xlabel('Number of components')
plt.ylabel('Total explained variance')
plt.xlim(0, 63)
plt.yticks(np.arange(0, 1.1, 0.1))
plt.axvline(21, c='b')
plt.axhline(0.9, c='r')
plt.show();
```

2.3 Кластеризация

В общей вербальной постановке задача кластеризации выглядит так: необходимо найти разбиение исходного множества объектов на непересекающиеся подмножества (кластеры) так, чтобы объекты внутри одного кластера обладали высоким сходством, а объекты из разных кластеров сильно различались [7].

Существует несколько разных алгоритмов решения задачи кластеризации. Ниже приведен набор алгоритмов из пакета *sklearn* и результат их работы (рис. 2.8) [8].

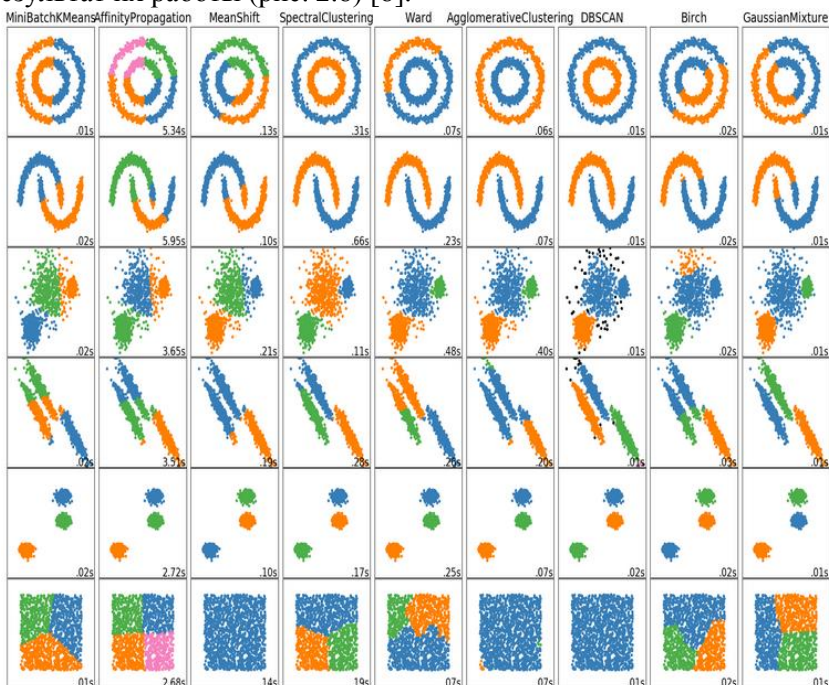


Рис 2.8. Результаты работы алгоритмов с разными типами данных.

По столбцам здесь представлены различные алгоритмы, а по строкам - типы данных. Изучая эти иллюстрации можно подобрать эффективные алгоритмы для решения своей задачи.

Поучимся работать с двумя алгоритмами кластеризации – иерархическим (агломеративным) и неиерархическим (*KMeans*).

Иерархический (агломеративный) алгоритм

#Импортируем библиотеку scipy: метод hierarchy для кластеризации и метод pdist для расчёта попарных расстояний

```
from scipy.cluster import hierarchy
from scipy.spatial.distance import pdist
```

#Случайно в интервалах зададим данные для кластеризации

```
X = np.zeros((150, 2))
```

```
np.random.seed(seed=42)
```

```
X[:50, 0] = np.random.normal(loc=0.0, scale=.3, size=50)
```

```
X[:50, 1] = np.random.normal(loc=0.0, scale=.3, size=50)
```

```
X[50:100, 0] = np.random.normal(loc=2.0, scale=.5, size=50)
```

```
X[50:100, 1] = np.random.normal(loc=-1.0, scale=.2, size=50)
```

```
X[100:150, 0] = np.random.normal(loc=-1.0, scale=.2, size=50)
```

```
X[100:150, 1] = np.random.normal(loc=2.0, scale=.5, size=50)
```

```
distance_mat = pdist(X)
```

pdist посчитает нам верхний треугольник матрицы попарных Евклидовых расстояний

```
Z = hierarchy.linkage(distance_mat, 'single')
```

#linkage – реализация агломеративного алгоритма (см. рис. 2.9)

```
plt.figure(figsize=(10, 5))
```

```
dn = hierarchy.dendrogram(Z, color_threshold=0.5)
```

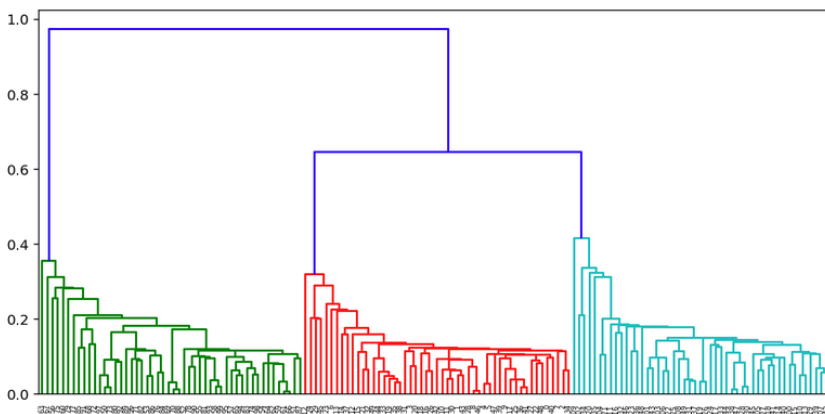


Рис. 2.9. Результат работы агломеративного алгоритма.

Метод *k*-средних (KMeans)

#Начнём с того, что зададим и отобразим на плоскости три кластера точек (рис. 2.10)

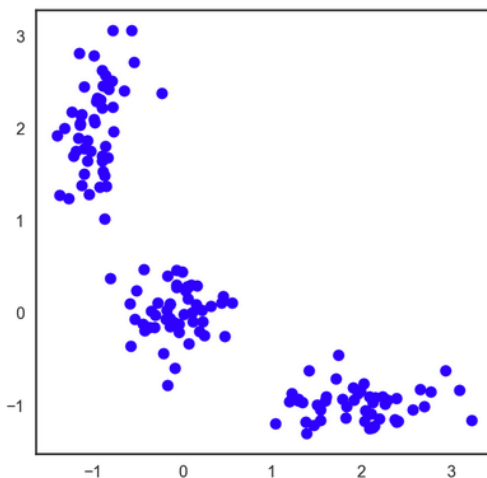


Рис. 2.10. Заданные нами три кластера точек.

```
Z = np.zeros((150, 2))
np.random.seed(seed=42)
Z[:50, 0] = np.random.normal(loc=0.0, scale=.3, size=50)
Z[:50, 1] = np.random.normal(loc=0.0, scale=.3, size=50)

Z[50:100, 0] = np.random.normal(loc=2.0, scale=.5, size=50)
Z[50:100, 1] = np.random.normal(loc=-1.0, scale=.2,
```



```

size=50)

Z[100:150, 0] = np.random.normal(loc=-1.0, scale=.2,
size=50)
Z[100:150, 1] = np.random.normal(loc=2.0, scale=.5, size=50)

plt.figure(figsize=(5, 5))
plt.plot(Z[:, 0], Z[:, 1], 'bo');
plt.show()

```

Как определить, сколько кластеров нужно выбрать для заданного датасета? Метрикой здесь является сумма всех расстояний (J) в кластерах (C_k). Зададим итерационную процедуру по подсчету J , увеличивая число кластеров на единицу, начиная с одного C_1 . С увеличением числа кластеров, $J(C_k)$ начнет уменьшаться. Будем выполнять эту процедуру до тех пор, пока $J(C_{k-1}) - J(C_k) \gg J(C_k) - J(C_{k+1})$, т.е. пока $J(C_k)$ на очередном шаге не будет меняться значительно слабее, чем на предыдущем. Тогда S_i и является показателем эффективного числа кластеров (рис. 2.11).

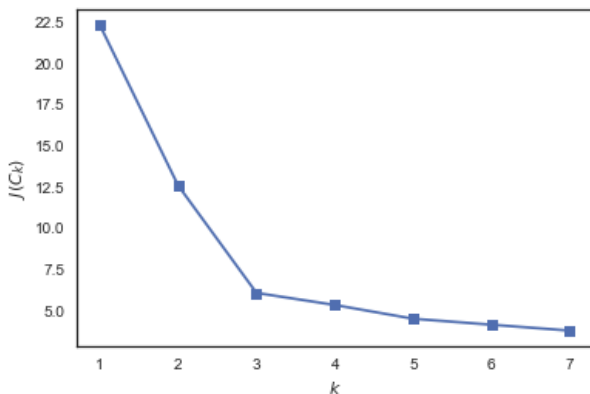


Рис. 2.11. График зависимости суммарного внутрикластерного расстояния от числа кластеров.

Из рисунка 2.11, где по горизонтали представлено число кластеров, а по вертикали – суммарное расстояние между всеми кластеризуемыми точками, видно, что наиболее предпочтительное количество кластеров равно 3.

```

from sklearn.cluster import Kmeans

inertia = []
for k in range(1, 8):
    kmeans = KMeans(n_clusters=k, random_state=1).fit(Z)

```

```

inertia.append(np.sqrt(kmeans.inertia_))

plt.plot(range(1, 8), inertia, marker='s');
plt.xlabel('$k$');
plt.ylabel('$J(C_k)$');
plt.show()

```

```

#строим модель с 3-мя кластерами
kmeans = KMeans(n_clusters=3)
kmeans.fit(Z)
#проставленные метки алгоритмом
dataTrainY=kmeans.labels_
#отображаем полученные кластеры на плоскости (рис. 2.12)

```

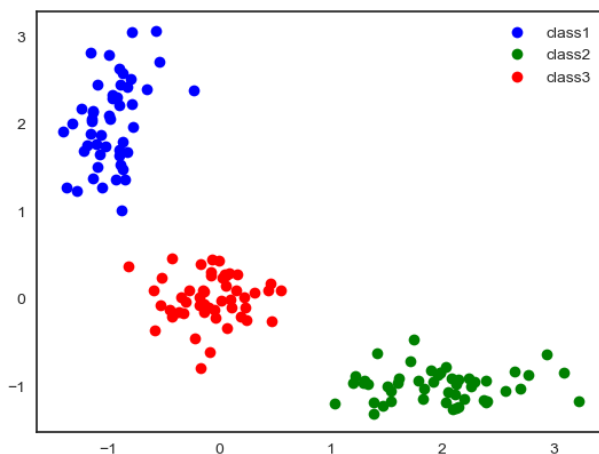


Рис. 2.12. Результат работы алгоритма кластеризации KMeans.

```

plt.plot(Z[dataTrainY==0,0],Z[dataTrainY==0,1], 'bo',
label='class1')
plt.plot(Z[dataTrainY==1,0],Z[dataTrainY==1,1], 'go',
label='class2')
plt.plot(Z[dataTrainY==2,0],Z[dataTrainY==2,1], 'ro',
label='class3')
plt.legend(loc=0)
plt.show()

```

2.4 Метрики качества кластеризации

Для измерения качества кластеризации существует несколько внешних и внутренних метрик качества. Внешние используют информацию об истинном разбиении на кластеры, в то время как внутренние метрики не используют никакой внешней информации, и оценивают качество кластеризации, основываясь только на наборе

кластеризуемых данных. Оптимальное число кластеров обычно определяют с использованием внутренних метрик.

Наиболее часто используют метрики *Adjusted Rand Index (ARI)* и *Adjusted Mutual Information (AMI)*, реализованные, например, в [*sklearn.metrics*](#) [9].

Для того чтобы воспользоваться метриками нужно подгрузить пакет *sklearn.metrics*

```
from sklearn import metrics
```

А далее в качестве параметра методов передать реальные метки класса, и те метки, которые проставил алгоритм.

Для *ARI* : `metrics.adjusted_rand_score_` (истинные метки, предсказанные метки *dataTrainY*).

Для *AMI* аналогичные параметры, только метод - `metrics.adjusted_mutual_info_score_`.

2.5 Домашнее задание

1. Загрузите выборку `load_digits()` и масштабируйте признаки с помощью `scale` из [*sklearn.preprocessing*](#)
2. Выведите размерность данных (`load_digits().data`), количество признаков и объектов, а так же количество уникальных значений в `load_digits().target`
3. Создайте *KMeans* из пакета *sklearn.cluster*, где в качестве параметров передайте `init='k-means++'`, `n_clusters=`количество уникальных значений из пункта 2, `n_init=10`, а в качестве данных масштабированные признаки.
4. Посчитайте значения следующих метрик: *ARI (Adjusted Rand Index)* и *AMI (Adjusted Mutual Information)*, а так же время работы алгоритма.
5. Аналогично проделайте для модели *KMeans*, где параметр `init='random'`
6. Примените метод *PCA*, задав количество компонент задайте равным количеству уникальных значений.
7. Снова создайте модель *KMeans*, где параметр `init` примет значения равные компонентам *PCA* (`pca.components_`).
8. Сравните все три подхода по времени и различными метрикам. Сделайте выводы и обоснуйте их.
9. Для наглядности отобразите данные на *2D* плоскости, так же отобразите центры каждого кластера и границы каждого кластера, используя любую из трёх моделей.

2.6 Контрольные вопросы

1. Для чего предназначен метод главных компонент и какова его основная идея?
2. Чем специфичны задачи обучения без учителя?
3. Как можно оценить качество кластеризации?
4. Какова постановка задачи кластеризации?
5. Если мы хотим кластеризовывать кольцевидные элементы какие алгоритмы лучше всего подойдут?
6. Какой процент остаточной дисперсии нужно выбрать для выбора n -ого количества главных компонент? На чём основывается этот выбор?
7. Какова основная идея алгоритма *KMeans*?
8. Какова основная идея агломеративного алгоритма?

3. Лабораторная работа № 3. Машинное обучение с учителем. Методы классификации (Classification) и регрессии (Regression)

Цель работы: изучение на практике методов машинного обучения с учителем для решения задач классификации и регрессии.

3.1 Машинное обучение с учителем

Обучение с учителем подразумевает предварительную разметку человеком объектов из обучающей выборки, т.е. присваивание этим объектам конечного числа меток для задач классификации или значений зависимой переменной для задач регрессии.

Тогда задача классификации представляется как отнесение объектов тестовой выборки к одной из ранее присвоенных меток, т.е. к одному из классов.

Под задачей регрессии подразумевают определение математической модели (функции регрессии), устанавливающей функциональную связь между зависимой переменной и группой независимых переменных с учетом ошибки модели.

Все массивы данных для обработки и тексты программ, реализующих методы классификации и регрессии, для данной лабораторной работы можно скачать по ссылке [https://github.com/tpo9hbi4/MachineDeepLearning/tree/master/ClassificationAndRegression\(4\)](https://github.com/tpo9hbi4/MachineDeepLearning/tree/master/ClassificationAndRegression(4)) .

3.2 Методы классификации

Рассмотрим на практике решение задачи классификации двумя различными методами – методом решающих деревьев [10] и методом ближайшего соседа [11].

Деревья принятий решений

```
# импортируем библиотеки
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# загрузим датасет ирисов
iris = load_iris()
X = iris.data
y = iris.target
X_train,X_test,y_train,y_test = train_test_split(X,y,
test_size=0.25, random_state = 241)
# построим модель, используя деревья принятий решений.
clf = tree.DecisionTreeClassifier()
clf.fit(X_train,y_train)
# предскажем данные
y_new = clf.predict(X_test)
# зафиксируем долю верных ответов
print(accuracy_score(y_test,y_new))
tree.export_graphviz(clf,out_file="treeClassification")
importances = clf.feature_importances_
# визуализируем наиболее важные признаки (см. рис.3.1)
print(importances)

0.8947368421052632
[0.03362152 0.          0.91454287 0.05183561]
```

Рис. 3.1. Наиболее важные признаки, полученные программой.

Задание:

Аналогично первой лабораторной работе вывести 2D изображение данных, где тремя разными цветами отметить реальные метки классов, а так же 2D изображение данных, где тремя цветами отметить метки классов, полученные алгоритмом деревьев принятий решений.

Метод ближайших соседей

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

iris = datasets.load_iris()

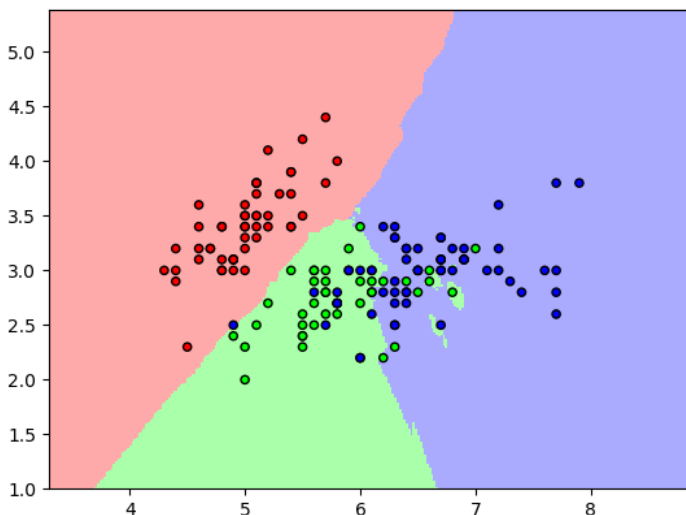
X = iris.data[:, :2]
y = iris.target

h = 0.02 # шаг
# создадим список цветов
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA',
                              '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00',
                              '#0000FF'])

# построим модель
clf = neighbors.KNeighborsClassifier(n_neighbors)
clf.fit(X, y)
# изобразим границы классов и реальные их элементы (рис.
3.2)
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                      np.arange(y_min, y_max, h))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
            edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```



*Рис. 3.2. Результат работы программы
(фон – границы классов, точки – элементы классов).*

3.3 Методы регрессии

Рассмотрим на практике решение задачи регрессии двумя различными методами – методом линейной регрессии [12] и методом ближайших соседей [13].

Линейная Регрессия

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Загрузим датасет
diabetes = datasets.load_diabetes()
print(diabetes)

# Выберем один признак
diabetes_X = diabetes.data[:, np.newaxis, 2]
print(diabetes_X)

# Выделим обучающие и тестовые признаки
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Выделим обучающие и тестовые ответы
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

```

# Создаём объект линейной регрессии
regr = linear_model.LinearRegression()
# обучим модель
regr.fit(diabetes_X_train, diabetes_y_train)
# Предскажем значение
diabetes_y_pred = regr.predict(diabetes_X_test)
# Выведем коэффициенты регрессии
print('Coefficients: \n', regr.coef_)
# Метрики оценки
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test,
                           diabetes_y_pred))
print('Variance score: %.2f' %
      r2_score(diabetes_y_test, diabetes_y_pred))
# Изобразим точки и аппроксимирующую прямую (рис. 3.3)
plt.scatter(diabetes_X_test, diabetes_y_test, col-
or='black')
plt.plot(diabetes_X_test, diabetes_y_pred, col-
or='blue', linewidth=3)
plt.xlabel('$data$')
plt.ylabel('$target$')
plt.show()

```

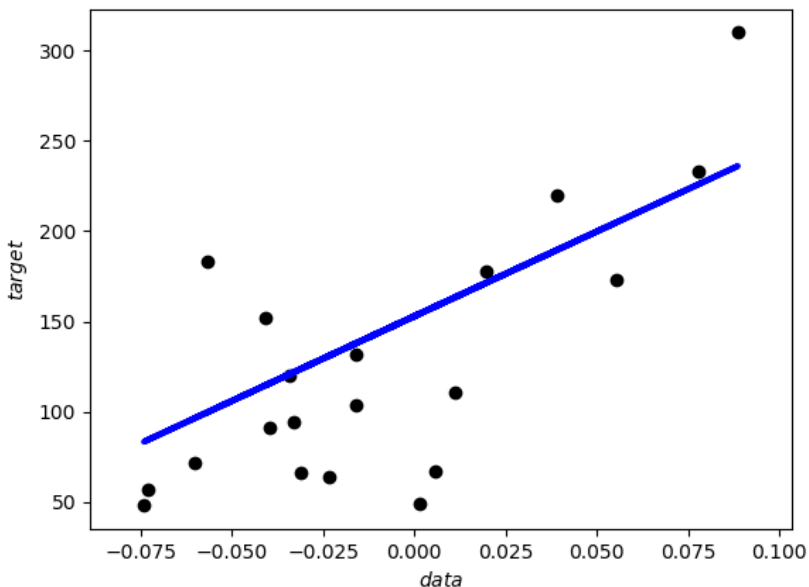


Рис. 3.3. Аппроксимирующая прямая точек датасета.

Метод ближайших соседей

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import neighbors
#Сгенерируем данные
np.random.seed(0)
X = np.sort(5 * np.random.rand(40, 1), axis=0)
print(X)
T = np.linspace(0, 5, 500)[: , np.newaxis]
print(T)
y = np.sin(X).ravel()
print(y)

y[::5] += 1 * (0.5 - np.random.rand(8))

n_neighbors = 5

knn = neighbors.KNeighborsRegressor(n_neighbors)
y_ = knn.fit(X, y).predict(T)

# Нарисуем это всё (рис. 3.4)
plt.subplot(2, 1, 1)
plt.scatter(X, y, c='k', label='data')
plt.plot(T, y_, c='g', label='prediction')
plt.axis('tight')
plt.legend()
plt.tight_layout()
plt.show()
```

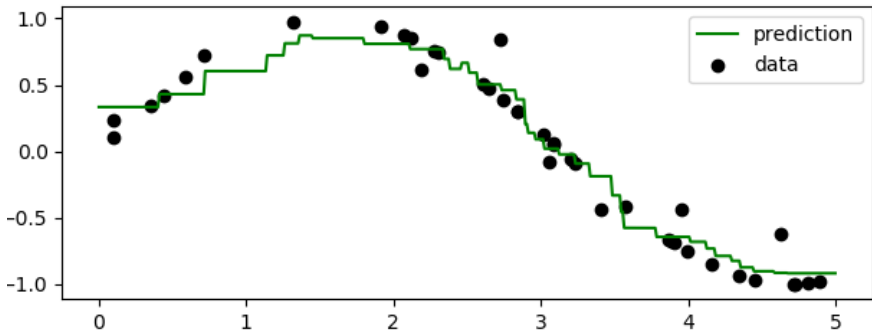


Рис. 3.4. Аппроксимирующая кривая точек датасета.

3.4 Домашние задания

I. Изучим данные о пассажирах Титаника. Решим такую задачу классификации: предсказать, кто выживет на корабле, зная различные реквизиты пассажиров.

Инструкция по выполнению

1. Загрузите выборку [ссылка на выборку titanic.csv \(https://github.com/tpo9hbi4/MachineDeepLearning\)](https://github.com/tpo9hbi4/MachineDeepLearning) с помощью функции `read_csv` из пакета `pandas`.
2. Отберите из выборки следующие признаки: `Pclass`, `Fare`, `Age` и `Sex`.
3. При присутствии строковых значений замените их на числовые.
4. Создайте столбец ответов — `Survived`.
5. Пропущенные значения в выборке нужно удалить. Для обнаружения пропущенных значений имеется функция `np.isnan`.
6. Используя `DecisionTreeClassifier` постройте модель, где все параметры будут использоваться по умолчанию, кроме параметра `random_state`, его значение задайте равным 100.
7. Найдите два самых важных признака. В ответе к задаче укажите их.

II. В этом задании следует найти оптимальное значение k для алгоритма k ближайших соседей. Решим следующую задачу классификации: необходимо предсказать сорт винограда, из которого сделано вино, зная его химические характеристики.

Инструкция по выполнению

1. Загрузите выборку [ссылка на выборку wine.csv \(https://github.com/tpo9hbi4/MachineDeepLearning\)](https://github.com/tpo9hbi4/MachineDeepLearning) с помощью функции `read_csv` из пакета `pandas`.
2. Разделите выборку на столбцы признаков и столбец ответов. Ответы записаны в 1-ом столбце, признаки — в столбцах со 2-ого до последнего. Подробнее о признаках по адресу <https://github.com/tpo9hbi4/MachineDeepLearning> файл `wine.names`.
3. Используя кросс-валидацию по 10 блокам оцените качество. Для этого нам понадобится метод `KFold` из пакета `sklearn.cross_validation`. Создайте разбиения таким образом, чтобы перемешать нашу выборку перед созданием блоков (установить значение `shuffle` - параметра равным `True`), и значением `random_state=100`.

4. Качество оцените с помощью метода *acurancy* для этого используйте *cross_val_score* из пакета *sklearn.cross_validation*.
5. Вычислите качество классификации для метода *k* - ближайших соседей, при изменении *k* от 1 до 100. Какому *k* соответствует лучшее качество и чему оно равно? *Это будет первый ответ на задание.*
6. Используя функции *sklearn.preprocessing.scale* масштабируйте признаки и снова вычислите лучшее *k* на кросс-валидации.
7. Какому *k* соответствует лучшее качество и чему оно равно для масштабированных признаков? *Это будет второй ответ на задание.* Улучшило ли это работу алгоритма?

III. Изучим набор данных *Boston*, и с помощью метода *k*-ближайших соседей (*sklearn.neighbors.KNeighborsRegressor*) решим задачу регрессии – предсказать стоимость жилья, зная различные его характеристики, подробно описанные здесь [https://github.com/tpo9hbi4/MachineDeepLearning/blob/master/ClassificationAndRegression\(4\)/Dataset/boston.info](https://github.com/tpo9hbi4/MachineDeepLearning/blob/master/ClassificationAndRegression(4)/Dataset/boston.info)

Инструкция по выполнению

1. Используя *sklearn.datasets.load_boston()* получим данные.
2. Масштабируйте признаки с помощью *sklearn.preprocessing.scale*.
3. Нужно перебрать значения *p* конструктора *KNeighborsRegressor* таким образом, чтобы оно изменялось от 1 до 20 и содержало 300 вариантов, также у конструктора задайте параметры *n_neighbors=6* и *weights='distance'* — данный параметр добавляет в алгоритм веса, зависящие от расстояния до ближайших соседей. В качестве метрики качества используйте среднеквадратичную ошибку (параметр *scoring='mean_squared_error'* у *cross_val_score*). Качество оценивайте, как и в пункте 3, 4 домашнего задания №2, используя кросс-валидацию по 10 блокам.
4. Вычислите лучшее *p*, когда оценка качества была наилучшей. Стоит помнить, что *cross_val_score* возвращает массив значений качества по блокам; необходимо найти максимальное значение среди средних всех значений показателей. *Это и будет ответ на задачу.*

3.5. Контрольные вопросы

1. Чем отличаются методы обучения с учителем и обучения без учителя?
2. Какие типы ответов в задачах обучения с учителем вы знаете?

3. На какой идее основан алгоритм деревьев решений?
4. На какой идее основан алгоритм ближайших соседей?
5. Какое существует общее уравнение для линейной регрессии для двумерного пространства
6. Чем отличается задачи регрессии и классификации?
7. Какие метрики оценки качества алгоритмов Вам известны?
8. Что означает операция нормализации признаков, и чем она отличается от операции стандартизации?

4. Лабораторная работа № 4. Рекомендательные системы.

Методы построения ассоциативных правил и коллаборативной фильтрации

Цель работы: изучение на практике основных методов, используемых при создании рекомендательных систем – методов построения ассоциативных правил по корзинам «покупателей» (*Sequential Pattern Mining - SPM*) и метода коллаборативной фильтрации (*Collaborative Filtering*) для формирования групп по интересам.

4.1 Персонализация онлайн-маркетинга

Задача рекомендательной системы – проинформировать пользователя о товаре или услуге, которые ему могут быть интересны в данный момент времени. Пользователь получает дополнительную информацию, а сервис зарабатывает на предоставлении качественных услуг. Услуги - это не обязательно прямые продажи, сервис также может зарабатывать на комиссионных или просто увеличивать лояльность пользователей, которая потом выливается в рекламные и иные доходы [14].

Персонализация онлайн-маркетинга - тренд последнего десятилетия. По оценкам *McKinsey*, 35% выручки *Amazon* или 75% *Netflix* приходится именно на рекомендованные товары. Методы, используемые в рекомендательных системах, могут анализировать корзины покупок, строя персональные ассоциативные правила, или могут анализировать таблицы предпочтений пользователей, чтобы заполнить пустые позиции в ней, и тем самым определить, что еще можно интересного предложить покупателям.

Все массивы данных для обработки и тексты программ, реализующих методы построения ассоциативных правил и коллаборатив-

ной фильтрации, для данной лабораторной работы можно скачать по ссылке

[https://github.com/tpo9hbi4/MachineDeepLearning/tree/master/RecommendationSystems\(3\)](https://github.com/tpo9hbi4/MachineDeepLearning/tree/master/RecommendationSystems(3)))

4.2 Поиск ассоциативных правил

Одной из задач при создании современных рекомендательных систем является задача поиска ассоциативных правил, позволяющих при анализе корзинок покупок рекомендовать очередные, вероятно необходимые покупателю, товары. Аналогично, при анализе последовательностей изученных курсов или запрошенных компьютерной программой массивов данных можно рекомендовать очередные курсы обучения в Coursera или массивы данных для обработки и анализа.

Для поиска подобных ассоциативных правил необходимо сначала находить последовательные шаблоны (*Sequential Pattern Mining*) – цепочки купленных товаров или запрошенных датасетов, и затем, исследуя их, строить устойчивые ассоциации.

К настоящему времени для решения этой задачи разработано несколько алгоритмов, с разной степенью эффективности анализирующих различные типы последовательностей товаров и услуг.

Одним из наиболее эффективных алгоритмов поиска ассоциативных правил является алгоритм ***FP-growth*** [15]. Его название можно перевести как «выращивание популярных (часто встречающихся) предметных наборов». Этот алгоритм позволяет не только избежать затратной процедуры генерации кандидатов, но и уменьшить необходимое число проходов по базе данных предметных наборов до двух.

В основе метода лежит преобработка базы транзакций, путем преобразования ее в компактную древовидную структуру, называемую *Frequent-Pattern Tree* – дерево популярных предметных наборов (откуда и название алгоритма). В дальнейшем для краткости будем называть эту структуру *FP-дерево* (рис. 4.1). К основным преимуществам данного метода относятся:

1. Сжатие БД транзакций в компактную структуру, что обеспечивает очень эффективное и полное извлечение частых предметных наборов;

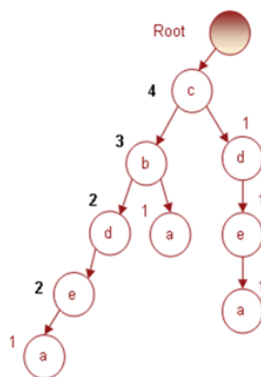


Рис.4.1. Дерево популярных

наборов FP-growth.

2. При построении FP-дерева используется технология разделения и властуй (лат. *divide et impera*), которая позволяет выполнить декомпозицию одной сложной задачи на множество более простых;

3. Позволяет избежать затратной процедуры генерации кандидатов, характерной, например, для алгоритма «*Apriori*».

Идея другого алгоритма *Apriori* базируется на двух относительных пороговых величинах - поддержки (S) и достоверности (D) и заключается в выполнении следующих шагов [16]:

Шаг 1. Среди всех исходных последовательностей элементов L выделяем множество L_1 таких одноэлементных наборов, поддержка которых больше некоторой заданной поддержки S .

Шаг 2. Формируем множество L_2 , путём составления всех возможных пар (двухэлементных наборов) из множества L_1 , и далее отбираем только те наборы, поддержка которых больше ранее заданного S .

Шаг 3. По аналогии с шагом 2 строим множества трехэлементных и т.д. наборов $L_3 \dots L_n$, до тех пор, пока множество L_n после отбора не станет пустым.

Шаг 4. Используя полученные многоэлементные наборы, начиная с более длинных L_{n-1} , и до L_2 , строим множество ассоциативных правил с достоверностью, превышающей заданное пороговое значение D .

Работа с алгоритмами FP-growth и Apriori

Замечание: Для работы с этими алгоритмами необходимо на рабочее место необходимо установить **Python** версии **2.7**.

Дальнейшие действия:

А) Скачаем архивы с предоставленными тестовыми файлами и готовыми реализациями алгоритмов из веб-сервиса *GitHub*.

Немного о выборках:

Имеем шесть выборок с разными параметрами (см. рис. 4.2):

- среднее число транзакций (T) для неоднородных баз или точное число транзакций для однородных баз (C) в клиентских последовательностях,

- среднее число предметов в транзакциях (I),

- количество клиентских последовательностей (D).

Б) Протестируем алгоритм FP-growth

Откроем командную строку. Перейдем в директорию, куда скачали папку с алгоритмом *FP-growth* при помощи команды:

```
cd \Путь
```

Далее для запуска программы в консоли укажем:

```
python -m fp_growth -s {minimum support} {path to CSV file}
```

Например:

```
python -m fp_growth -s 4 examples/tsk.csv
```

Type	T	C	I	D
Dataset1	-	10	10	1 000
Dataset2	-	20	20	10 000
Dataset3	-	73	73	10 000
Dataset4	25	-	10	10 000
Dataset5	10	-	4	100 000
Dataset6	40	-	10	100 000

Рис. 4.2. Пример обрабатываемых выборок.

В консоли увидим результат работы программы и время ее работы. Изменяя значения минимальной поддержки, получим разные значения времени работы программы. Эти данные нужно занести в таблицу, например, в такую, как на рисунке 4.3.

dataset1=C10D1k	FP-Growth
Поддержка1%	0,719
Поддержка5%	0,097
Поддержка10%	0,049
Поддержка20%	0,016
Поддержка30%	0,009
Поддержка40%	0,008
Поддержка50%	0,007
Поддержка60%	0,008
Поддержка70%	0,007
Поддержка80%	0,007
Поддержка90%	0,007

Рис. 4.3. Время работы алгоритма FPGrowth.

Далее изменим минимальные поддержку (*MinSupp*) и достоверность (*MinConf*) и получим результаты для их разных комбинаций.

В) Протестируем алгоритм Apriori

Аналогично пункту Б) для тестирования алгоритма *Apriori* в консоли используем следующую команду:

python apriori.py -f файл_c_выборкой.csv -s MinSupp -c MinConf

Файл с выборкой имеет формат, представленный на рис. 4.4, где через запятую перечисляются используемые наборы данных и, соответственно, номер строки – *UserID*.

1	20,156,158,189,204,209,222,239,270,279,282,285,288,292,300,310,315,320,336,383,
2	20,156,159,189,204,210,230,239,270,278,282,285,288,292,300,310,315,320,346,385,
3	20,156,159,189,204,210,230,239,270,278,282,285,288,292,300,310,315,320,346,385,
4	20,156,158,189,203,211,213,253,269,278,282,285,288,293,300,310,315,320,321,382,
5	20,156,160,189,203,209,220,239,269,278,282,285,288,293,300,310,315,320,335,383,
6	20,156,160,189,203,209,220,239,269,278,282,285,288,293,300,310,315,320,335,383,
7	20,156,160,189,203,209,220,239,269,278,282,285,288,293,300,310,315,320,335,383,
8	20,156,161,189,207,209,230,239,271,279,282,285,288,294,300,310,315,320,345,383,

Рис. 4.4. Пример выборки для алгоритма Apriori.

Работа прототипа программы представлена на рис. 4.5. Порой результат занимает слишком много места, поэтому командой

```
> test1.txt
```

можно перенаправить вывод в файл, содержащийся в той же папке, что и исполняемый файл.

```
D:\Учёба\8 семестр\VKP\Практика\Apriori-master>python apriori.py -f c10d1k.csv -s 0.9 -c 1.1
2017-06-07 22:40:24.542000
2017-06-07 22:40:24.559000
0:00:00.017000
item: ('156',) , 0.999
item: ('20',) , 0.999
item: ('20', '156') , 0.999
```

Рис. 4.5. Работа прототипа программы.

Результаты тестирования алгоритма на выборке *Dataset1* необходимо представить в виде таблицы (пример ее представлен на рис. 4.6) и по ним построить график (пример - на рис. 4.7).

dataset1=C10D1k	
	Apriori
Поддержка1%	52,685
Поддержка5%	1,316
Поддержка10%	0,376
Поддержка20%	0,066
Поддержка30%	0,028
Поддержка40%	0,023
Поддержка50%	0,02
Поддержка60%	0,02
Поддержка70%	0,019
Поддержка80%	0,017
Поддержка90%	0,017

Рис. 4.6. Результат работы Apriori на выборке Dataset1.

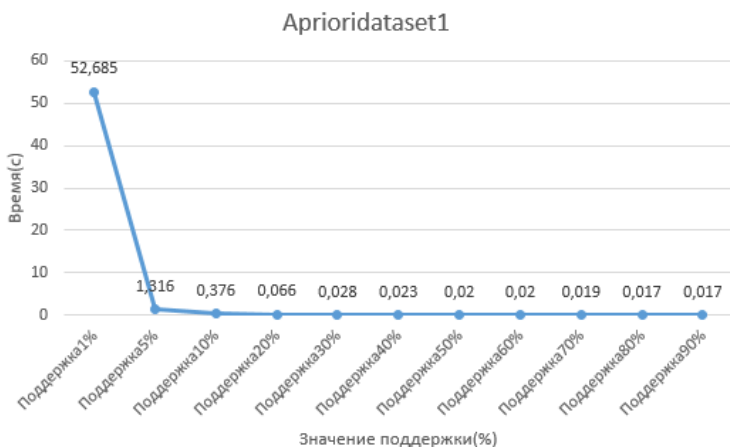


Рис. 4.7. График времени работы Apriori на выборке Dataset1.

4.3 Методы коллаборативной фильтрации

Другой задачей при создании современных рекомендательных систем является задача коллаборативной фильтрации, решение которой позволяет заполнять предполагаемыми оценками таблицу оценок A (рис. 4.8) услуг или продуктов (*item*, i), которые дали несколько (далеко не все) покупатели (*user*, u).

В этой лабораторной работе для решения задачи коллаборативной фильтрации будем использовать метод сингулярного разложения матрицы (*Singular Value Decomposition*, *SVD*) [14].

	i_1	i_2	i_3	...	i_m
U_1	-	4	-	...	5
U_2	4	-	2	...	-
U_3	-	5	-	...	3
...
U_n	4	-	2	...	-

Рис. 4.8. Пример таблицы оценок A .

Как следует из названия, в основе метода лежит предположение, что исходная таблица оценок A (размерности $n \times m$) является матрицей (хотя строго говоря – это не так), которую в соответствии с теоремой о сингулярном разложении всегда можно представить произведением трех матриц $A = U \times \Sigma \times V^T$ с размерностями $n \times n$;

$n \times m$ и $m \times n$, соответственно. Здесь матрицы U и V – ортогональные, а Σ – диагональная. Далее, если занулить часть нижних диагональных элементов, начиная с последнего, матрицы Σ , то получим эту матрицу меньшей размерности. Обозначим новую размерность матрицы Σ как $f \times f$, где $f < n$ или даже $f \ll n$. Теперь при перемножении трех матриц мы получим хорошее низкоранговое приближение A' исходной матрицы A :

$$A' = U' \times \Sigma' \times V'^T \approx A \quad (1).$$

$n \times m \quad n \times f \quad f \times f \quad f \times m \quad n \times m$

И, перемножив две первые матрицы в выражении (1), получим вместо них одну (U''), тогда, A' будет равно произведению (2) двух матриц (рис. 4.9):

$$A' = U'' \times V'^T \approx A \quad (2).$$

$n \times m \quad n \times f \quad f \times m \quad n \times m$

которые нам нужно обучить, т.е. вычислить и проставить значения $r_{i,j}$ и $q_{j,k}$ так, чтобы суммарное среднеквадратичное отклонение оценок, предполагаемых данным методом и ранее предоставленных пользователями в таблице A , было минимальным.

U''				V'^T					
	r_1	...	r_f	1	i_1	i_2	i_3	...	i_m
U_1	$r_{1,1}$...	r_{1f}	q_1	$q_{1,1}$	$q_{1,2}$	$q_{1,3}$...	$q_{1,m}$
U_2	$r_{2,1}$...	r_{2f}	q_2	$q_{2,1}$	$q_{2,2}$	$q_{2,3}$...	$q_{2,m}$
U_3	$r_{3,1}$...	r_{3f}
...	q_f	$q_{f,1}$	$q_{f,2}$	$q_{f,3}$...	$q_{f,m}$
U_{n-1}	$r_{n-1,1}$...	$r_{n-1,f}$						
U_n	$r_{n,1}$...	r_{nf}						

Рис. 4.9. Обучаемые матрицы методом SVD.

A) Скачиваем архивы с предоставленными тестовыми файлами и готовой реализации алгоритма из веб-сервиса *GitHub* по ссылке [https://github.com/tpo9hbi4/MachineDeepLearning/tree/master/RecommendationSystems\(3\)](https://github.com/tpo9hbi4/MachineDeepLearning/tree/master/RecommendationSystems(3))

B) Тестируем алгоритм SVD, решая классическую задачу поиска среди всех зарегистрированных в фильмотеке пользователей таких, которые имеют близкие оценки предлагаемых к просмотру фильмов [17]. Исходно имеем две таблицы:

- характеристик фильмов в фильмотеке (*df_movies*);
- некоторых оценок (рейтингов) фильмов, просмотренных зарегистрированными пользователями (*df_ratings*);

стрированными пользователями фильмотеки (*df_rates*).

```
import pandas as pd

#Загрузим данные с выборкой
df_rates=pd.read_csv('D:\Dataset\SVD\userRatedmovies.dat',sep='\t')
df_movies=pd.read_csv("D:\Dataset\SVD\movies.dat",sep='\t',encoding='iso-8859-1')

#Посмотрим содержимое данных (рис. 4.10, 4.11)
print(df_rates.head())

from sklearn.preprocessing import LabelEncoder

#Посмотрим сколько пользователей, с какими ID есть в выборке (рис. 4.12)
print(df_rates.userID.min(),df_rates.userID.max())
print(len(df_rates.userID.unique()))
print(df_rates.userID.shape)

enc_user=LabelEncoder()
enc_mov=LabelEncoder()
```

	userID	movieID	rating	date_day	date_month	date_year	date_hour	\
0	75	3	1.0	29	10	2006	23	
1	75	32	4.5	29	10	2006	23	
2	75	110	4.0	29	10	2006	23	
3	75	160	2.0	29	10	2006	23	
4	75	163	4.0	29	10	2006	23	

	date_minute	date_second
0	17	16
1	23	44
2	30	8
3	16	52
4	29	30

Рис. 4.10. Исходная таблица рейтингов.

```
print(df_movies.head())
```

	id	title	imdbID	\
0	1	Toy story	114709	
1	2	Jumanji	113497	
2	3	Grumpy Old Men	107050	
3	4	Waiting to Exhale	114885	
4	5	Father of the Bride Part II	113041	

	spanishTitle	\
0	Toy story (juguetes)	
1	Jumanji	
2	Dos viejos gruñones	
3	Esperando un respiro	

Рис. 4.11. Исходная таблица характеристик фильмов.

```

75 71534
2113
(855598,)

```

Рис. 4.12. Оценки пользователей фильмотеки.

```

enc_user=enc_user.fit(df_rates.userID.values)
enc_mov=enc_mov.fit(df_rates.movieID.values)

```

*#Отберём в таблице с фильмами лишь те, за которые
голосовали пользователи*

```

idx=df_movies.loc[:, 'id'].isin(df_rates.movieID)
df_movies=df_movies.loc[idx]

```

#Применили LabelEncoder для удобства

```

df_rates.loc[:, 'userID'] =
enc_user.transform(df_rates.loc[:, 'userID'].values)
df_rates.loc[:, 'movieID'] =
enc_mov.transform(df_rates.loc[:, 'movieID'].values)
df_movies.loc[:, 'id'] =
enc_mov.transform(df_movies.loc[:, 'id'].values)
print(df_rates.head())

```

#Матрица схожести

```

from scipy.sparse import coo_matrix
R = coo_matrix((df_rates.rating.values,
(df_rates.userID.values, df_rates.movieID.values)))
print(R.toarray())
print(R.toarray().shape)

```

```

#метод SVD
from scipy.sparse.linalg import svds
u,s,vt = svds(R,k=6)
print(u.shape)
print(s.shape)
print(vt.shape)
print(vt.T)

#Оценим методом ближайших соседей
from sklearn.neighbors import NearestNeighbors

nn=NearestNeighbors(n_neighbors=10)
v=vt.T
nn.fit(v)
_,ind = nn.kneighbors(v, n_neighbors=10)

#Просмотрим матрицу с ближайшими по схожести фильмами
print(ind[:10])

movie_titles=df_movies.sort_values('id').loc[:,'title'].
values
cols=['movie']+['nn_{}'.format(i) for i in
range(1,10)]
print(cols)
df_ind_nn=pd.DataFrame(data=movie_titles[ind],columns=
cols)
print(df_ind_nn.head())

#Выведем ближайшие фильмы к Терминатору
idx=df_ind_nn.movie.str.contains('Terminator')

print(df_ind_nn.loc[idx].head())

```

4.4. Домашние задания

I. (по методам *FP-growth* и *Apriori*)

1. Выполнить алгоритм *Apriori* на разных выборках. Оценить результат. Поддержку стоит выбирать разумно, если алгоритм работает более 5 минут, стоит остановиться на уменьшении значения поддержки.

2. Выполнить алгоритм *FP-growth* на разных выборках. Оценить результат. Поддержку стоит выбирать разумно, если алгоритм

работает более 5 минут, стоит остановиться на уменьшении значения поддержки.

3. Результаты по времени работы алгоритмов занести в ниже приведенную таблицу

<i>Dataset (номер) = C10D1k (у вас будет 6 разных)</i>		
<i>Поддержка</i>	<i>Время Apriori</i>	<i>Время FP-growth</i>
Поддержка 1%		
Поддержка 5%		
Поддержка 10%		
Поддержка 20%		
Поддержка 30%		
Поддержка 40%		
Поддержка 50%		
Поддержка 60%		
Поддержка 70%		
Поддержка 80%		
Поддержка 90%		

4. Заполнить аналогичные таблицы для всех шести выборок, и по шести таблицам построить шесть графиков зависимости, где по горизонтали отложить Поддержку (%), а по вертикали $Ln(t)$, где t – количество секунд, которое работал алгоритм.

II. (по методу SVD)

1. Используя программу, реализующую метод SVD приведенную выше в п. 4.3, необходимо найти, и распечатать наиболее близкие фильмы к фильму «Терминатор 2».

2. Найти ближайших пользователей по схожим интересам к пользователю №1.

А) Для этого понадобится работать с матрицей рейтингов R .

Б) По заданной матрице R рассчитайте косинусное расстояние с помощью метода `sklearn.metrics.pairwise.cosine_similarity` и выведите её размерность.

В) Скорректируйте меру схожести. Для этого нужно написать функцию, которая будет принимать на вход рейтинги пользователей u и v . Сначала требуется найти пересечение между ними. Если оно не нулевое, то нужно подсчитать схожесть, как косинус между соответствующими векторами, и для получения величины схожести - взять его с минусом и добавить единицу (+1).

Г) Чтобы подсчитать косинусное расстояние используйте метод `pdist`, где на вход ему передайте матрицу схожести (т.е. `R.toarray()`),

а в качестве метрики передайте написанную функцию в подпункте В).

Д) Выведите размерность полученного вектора. Чтобы получить стандартный вид, прогоните его через [squareform](#), получив на выходе нужную матрицу, размером 2113 x 2113 (по числу зарегистрированных пользователей в фильмотеке датасета).

4.5. Контрольные вопросы

по методам построения ассоциативных правил

1. В чём идея алгоритма *FP-growth*?
2. Важен ли порядок предметов в транзакции для алгоритма *FP-growth*?
3. В чём идея алгоритма *Apriori*?
4. Важен ли порядок предметов в транзакции для алгоритма *Apriori*?
5. В чём сходство и отличие алгоритмов *FP-growth* и *Apriori*?
6. С какими типами структур выборки могут работать алгоритмы *Apriori* и *FP-growth*?
7. Как загружать программу из файла *GitHub*?
8. С помощью какой функции можно измерять время работы программы?
9. Что за параметр *Minsupport* в задачах поиска ассоциативных правил?
10. На каком из файлов будет работать лучше *FP-growth*?
11. На каком из файлов будет работать лучше *Apriori*?
12. В каких случаях корректно сравнение алгоритмов?

по методу *SVD*

13. В какие типы матриц раскладывается таблица оценок A ?
14. Что обозначает каждая из этих матриц?
15. Как рассчитать среднеквадратичное отклонение ошибки в методе *SVD*?
16. К какому типу задач относят задачу коллаборативной фильтрации?
17. Какова основная идея обучения в методе *SVD*?

Список литературы

(жирным шрифтом выделена рекомендованная литература)

1. Что такое *data-driven*? И как вашей компании стать дата-ориентированной? *Data Review*, 08.11.2016.

<http://datareview.info/article/chto-takoe-data-driven-i-kak-vashej-kompanii-stat-data-orientirovannoj/>

2. Актуальность данных и аналитических исследований. Что такое аналитика? Виды анализа. *Business Analysis*.

<https://iiba.ru/aktualnost-dannyh-i-analiticheskikh-issledovanij-chto-takoe-analitika-vidy-analiza/>

3. *Kasey Panetta. Gartner Top 10 Strategic Technology Trends for 2019. Smarter With Gartner, October 15, 2018.*

<https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2019/>

4. *Augmented Analytics: what does it really mean? Mode Finance*, 2018-12-06.

<https://www.modefinance.com/en/blog/2018-12-06-augmented-analytics-what-does-it-really-mean->

5. **Орлов А.И. Прикладная статистика.** Электронный учебник. М.: Издательство «Экзамен», 2004.

http://www.aup.ru/books/m163/3_2_6.htm

6. Сушков А. Машинное обучение: от Ирисов до Телекома. *Habr*, 2017 <https://habr.com/ru/company/billing/blog/334738/>

7. **Воронцов К.В. Лекции по алгоритмам кластеризации и многомерного шкалирования**, 2007.

<http://www.ccas.ru/voron/download/Clustering.pdf>

8. *Scikit-learn. Machine Learning in Python. 2.3.1. Overview of clustering methods.*

<https://scikit-learn.org/stable/modules/clustering.html>

9. *Scikit-learn. Machine Learning in Python. 2.3.9. Clustering performance evaluation.*

<https://scikit-learn.org/stable/modules/clustering.html>

10. **Метод деревьев решений** для задачи классификации, – *DOCPlayer*, 2014.

<https://docplayer.ru/46578177-Metod-derevev-resheniy-dlya-zadachi-klassifikacii.html>

11. Метод *K*-ближайших соседей для решения задачи классификации, – *DOCPlayer*, 2014. <https://docplayer.ru/37869564-Metod-k-blizhayshih-sosedey-dlya-resheniya-zadachi-klassifikacii.html>

12. Воронцов К.В. Лекции по алгоритмам восстановления регрессии, 2007. <http://www.ccas.ru/voron/download/Regression.pdf>

13. *Tavish Srivastava. Introduction to k-Nearest Neighbors: Simplified* (with implementation in Python), Analytics Vidhya, 2018. <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>

14. Коточигов К. Анатомия рекомендательных систем. Части 1 и 2. ГК Ланит, *Habr*, 2018.
<https://habr.com/ru/company/lanit/blog/420499/>
<https://habr.com/ru/company/lanit/blog/421401/2>.

15. Орешков В. *FPG* - альтернативный алгоритм поиска ассоциативных правил. *Base Group Labs* (технологии анализа данных). <https://basegroup.ru/community/articles/fpg>

16. *Agrawal R., Srikant R. Mining Sequential Patterns. In Proc. of the 11th Int'l Conference on Data Engineering*, 1995.
<https://pdfs.semanticscholar.org/d6a0/e0b04a020ac6422b98b8e63027a6178060fd.pdf>

17. Сайт *Internet Movie Database (IMDb)* компании Amazon.com.Inc. <https://www.imdb.com>