Лабораторная работа №29

Обучение без учителя

Вторая группа алгоритмов машинного обучения, которую мы будем рассматривать, – это машинное обучение без учителя. Машинное обучение без учителя включает в себя все виды машинного обучения, когда ответ неизвестен и отсутствует учитель, указывающий ответ алгоритму. В машинном обучении без учителя есть лишь входные данные и алгоритму необходимо извлечь знания из этих данных.

Типы машинного обучения без учителя

В этой главе мы рассмотрим два вида машинного обучения без учителя: преобразования данных и кластеризацию.

Неконтролируемые преобразования (**unsupervised transformations**) – это алгоритмы, создающие новое представление данных, которое в отличие от исходного представления человеку или алгоритму машинного обучения будет обработать легче. Общераспространенное применение неконтролируемых преобразований – сокращение размерности. Мы берем высокоразмерное представление данных, состоящее из множества признаков, и находим новый способ представления этих данных, обобщая основные характеристики и получая меньшее количество признаков. Общераспространенное применение сокращения размерности – получение двумерного пространства в целях визуализации.

Еще одно применение неконтролируемых преобразований – поиск компонент, из которых «состоят» данные. Примером такого преобразования является выделение тем из коллекций текстовых документов. Здесь задача состоит в том, чтобы найти неизвестные темы, обсуждаемые в коллекции документов, а также выяснить, какие темы встречаются в каждом документе. Это может быть полезно для отслеживания в социальных сетях обсуждений таких тем, как выборы, контроль огнестрельного оружия или жизнь поп-звезд.

С другой стороны, алгоритмы кластеризации (**clustering algorithms**) разбивают данные на отдельные группы схожих между собой элементов. Рассмотрим пример загрузки фотографий в социальной сети. Часто вы формируете запросы типа «покажите мне все фотографии, на которых изображен Иван Петров». Для выполнения подобных запросов, администрация сайта, возможно, захочет сгруппировать фотографии, на которых изображен один и тот же человек. Однако при этом неизвестно, на каких загружаемых фотографиях кто показан, и неизвестно, какое количество различных пользователей присутствует на ваших фотографиях. Разумный подход заключался бы в том, чтобы извлечь все лица и разделить их на группы лиц, которые схожи между собой. Будем надеяться, что они соответствуют одному и тому же человеку и изображения в сгруппированном виде будут предъявлены вам.

Проблемы машинного обучения без учителя

Главная проблема машинного обучения без учителя – оценка полезности информации, извлеченной алгоритмом. Алгоритмы машинного обучения без учителя, как правило, применяются к данным, которые не содержат никаких меток, таким образом, мы не знаем, каким должен быть правильный ответ. Поэтому очень трудно судить о качестве работы модели. Например, наш гипотетический алгоритм кластеризации мог бы сгруппировать вместе все фотографии лиц в профиль и все фотографии лиц в анфас. Перед нами, несомненно, один из способов разбить коллекцию фотографий лиц на группы, но это совсем не то, что нам нужно. Тем не менее у нас нет никакой возможности «рассказать» алгоритму, что мы ищем, и часто единственный способ оценить результат работы алгоритма машинного обучения без учителя – ручная проверка этого результата.

Как следствие, алгоритмы машинного обучения без учителя часто используются в разведочных целях, когда специалист хочет лучше изучить сами данные. Еще одно общераспространенное применение алгоритмов машинного обучения без учителя заключается в том, что они служат этапом предварительной обработки данных для алгоритмов машинного обучения с учителем. Изучение нового представления данных иногда может повысить правильность алгоритмов машинного обучения с учителем или может привести к снижению времени вычислений и потребления объема памяти.

Прежде чем начать знакомство с «реальными» алгоритмами машинного обучения без учителя, мы кратко рассмотрим некоторые простые методы предварительной обработки данных, которые часто могут пригодиться. Хотя предварительная обработка данных и масштабирование часто применяются вместе с алгоритмами контролируемого обучения, методы масштабирования не используют учителя, что делает их методами неконтролируемого обучения.

Предварительная обработка и масштабирование

В предыдущей главе мы видели, что некоторые алгоритмы, например, нейронные сети и SVM, очень чувствительны к масштабированию данных. Поэтому обычной практикой является преобразование признаков с тем, чтобы итоговое представление данных было более подходящим для использования вышеупомянутых алгоритмов. Часто достаточно простого масштабирования признаков и корректировки данных. Программный код (рис. 1) показывает простой пример:

```
mglearn.plots.plot_scaling()
plt.show()
```

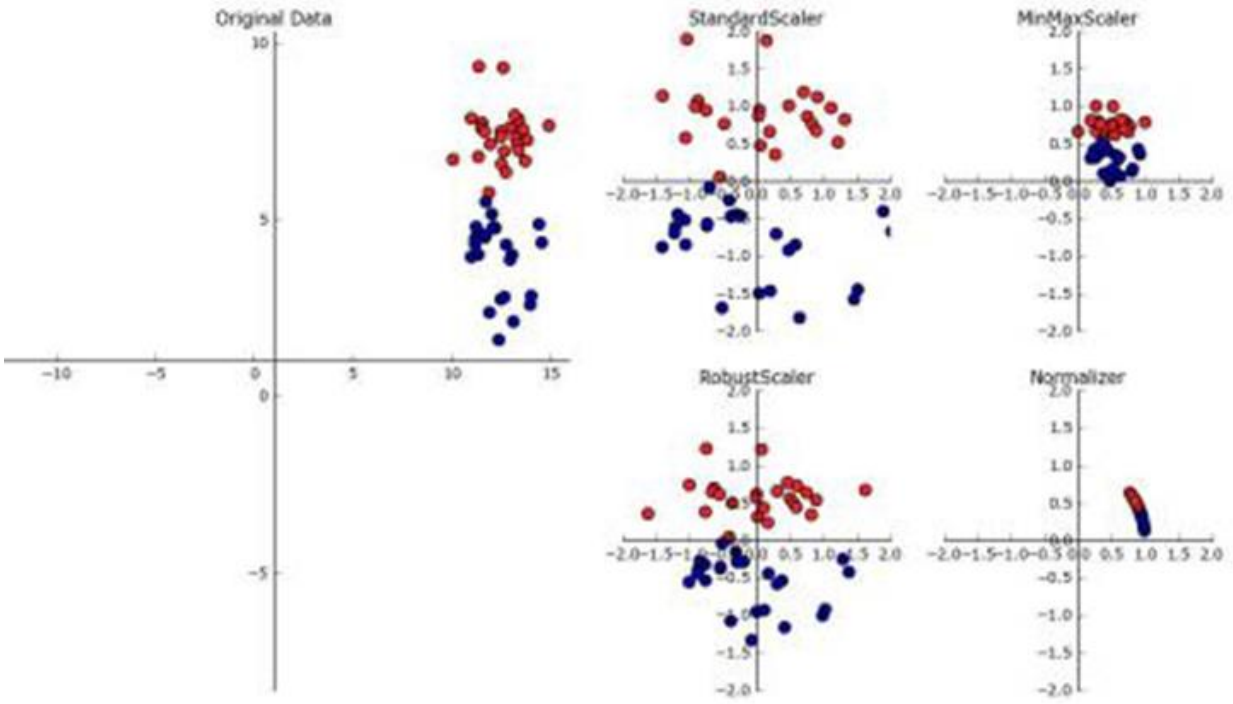

Рис. 1 Различные способы масштабирования и предварительной обработки данных

Различные виды предварительной обработки

Первый график на рис. 1 соответствует синтетическому двуклассовому набору данных с двумя признаками. Первый признак (ось x) принимает значения в диапазоне от 10 до 15. Второй признак (ось y) принимает значения примерно в диапазоне от 1 до 9.

Следующие четыре графика показывают четыре различных способа преобразования данных, которые дают более стандартные диапазоны значений. Применение **StandardScaler** в scikit-learn гарантирует, что для каждого признака среднее будет равно 0, а дисперсия будет равна 1, в результате чего все признаки будут иметь один и тот же масштаб. Однако это масштабирование не гарантирует получение каких-то конкретных минимальных и максимальных значений признаков. **RobustScaler** аналогичен **StandardScaler** в том плане, что в результате его применения признаки будут иметь один и тот же масштаб. Однако **RobustScaler** вместо среднего и дисперсии использует медиану и квартили. Это позволяет **RobustScaler** игнорировать точки данных, которые сильно отличаются от остальных (например, ошибки измерений). Эти странные точки данных еще называются выбросами (**outliers**) и могут стать проблемой для остальных методов масштабирования.

С другой стороны, **MinMaxScaler** сдвигает данные таким образом, что все признаки находились строго в диапазоне от 0 до 1. Для двумерного набора данных это означает, что все данные помещаются в прямоугольник, образованный осью x с диапазоном значений от 0 и 1 и осью y с диапазоном значений от 0 и 1.

И, наконец, **Normalizer** осуществляет совершенно иной вид масштабирования. Он масштабирует каждую точку данных таким образом, чтобы вектор признаков имел евклидову длину 1. Другими словами, он проецирует точку данных на окружность с радиусом 1 (или сферу в случае большого числа измерений). Вектор умножается на инверсию своей длины. Подобная нормализация используется тогда, когда важным является направление (но не длина) вектора признаков.

Применение преобразований данных

Теперь, когда мы увидели, что делают различные виды преобразований, давайте применим их, воспользовавшись scikit-learn. Мы будем использовать набор данных cancer. Методы предварительной обработки обычно применяются перед использованием алгоритма машинного обучения с учителем. Допустим, в качестве примера нам нужно применить ядерный SVM (SVC) к набору данных cancer и использовать **MinMaxScaler** для предварительной обработки данных. Мы начнем с того, что загрузим наш набор данных и разобьем его на тренировочный и тестовый наборы (обучающий и тестовый наборы нам нужны для оценки качества модели, которую мы построим с помощью алгоритма контролируемого обучения после предварительной обработки):

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
 random_state=1)
print(X_train.shape)
print(X_test.shape)
```

```
(426, 30)
(143, 30)
```

Напомним, что набор содержит 569 точек данных, каждая из которых представлена 30 признаками. Мы разбиваем набор данных на 426 примеров в обучающей выборке и 143 примера в тестовой выборке.

Как и в случае с моделями контролируемого обучения, построенными ранее, мы сначала импортируем класс, который осуществляет предварительную обработку, а затем создаем его экземпляр:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

Затем с помощью метода **fit** мы подгоняем **scaler** на обучающих данных. Для **MinMaxScaler** метод **fit** вычисляет минимальное и максимальное значения каждого признака на обучающем наборе. В отличие от классификаторов и регрессоров при вызове метода **fit scaler** работает с данными (X_train), а ответы (y_train) не используются:

```
scaler.fit(X_train)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

Чтобы применить преобразование, которое мы только что подогнали, то есть фактически отмасштабировать (scale) обучающие данные, мы воспользуемся методом **transform**. Метод **transform** используется в scikit-learn, когда модель возвращает новое представление данных:

```
min_on_training = X_train.min(axis=0)
range_on_training = (X_train - min_on_training).max(axis=0)
X_train_scaled = (X_train - min_on_training) / range_on_training
print("форма преобразованного массива: {}"
    .format(X_train_scaled.shape)
)
print("min значение признака до масштабирования:\n {}"
    .format(X_train.min(axis=0))
)
print("max значение признака до масштабирования:\n {}"
    .format(X_train.max(axis=0))
)
print("min значение признака после масштабирования:\n {}"
    .format( X_train_scaled.min(axis=0))
)
print("max значение признака после масштабирования:\n {}"
    .format( X_train_scaled.max(axis=0))
)
```

```
форма преобразованного массива: (426, 30)
форма преобразованного массива: (426, 30)
min значение признака до масштабирования: [
    6.98    9.71        43.79   143.50  0.05    0.02
    0.      0.  0.11    0.05    0.12    0.36    0.76
    6.80    0.          0.      0.      0.      0.01
    0.      7.93        12.02   50.41   185.20  0.07
    0.03    0.          0.      0.16    0.06
]
max значение    признака до масштабирования: [
    28.11       39.28       188.5       2501.0      0.16
    0.29        0.43        0.2         0.300       0.100
    2.87        4.88        21.98       542.20      0.03
    0.14        0.400       0.050       0.06        0.03
    36.04       49.54       251.20      4254.00     0.220
    0.940       1.17        0.29        0.58        0.15
]
min значение признака после масштабирования: [
    0.  0.  0.  0.  0.  0.
    0.  0.  0.  0.  0.  0.
    0.  0.  0.  0.  0.  0.
```

```
    0.  0.  0.  0.  0.  0.
    0.  0.  0.  0.  0.  0.
]
max значение признака после масштабирования: [
    1.  1.  1.  1.  1.  1.
    1.  1.  1.  1.  1.  1.
    1.  1.  1.  1.  1.  1.
    1.  1.  1.  1.  1.  1.
    1.  1.  1.  1.  1.  1.
]
```

Преобразованные данные имеют такую же форму, что и исходные данные – признаки просто смещены и масштабированы. Видно, что теперь все признаки принимают значения в диапазоне от 0 до 1, как нам и требовалось.

Чтобы применить SVM к масштабированным данным, мы должны преобразовать еще тестовый набор. Это снова делается с помощью вызова метода **transform**, на этот раз для X_test:

```
X_test_scaled = (X_test - min_on_training) / range_on_training
print("min значение признака после масштабирования:\n{}"
    .format(X_test_scaled.min(axis=0))
)
print("max значение признака после масштабирования:\n{}"
    .format(X_test_scaled.max(axis=0))
)
```

```
min значение признака после масштабирования:
[ 0.034 0.023   0.031   0.011   0.141   0.044   0.  0.  0.154   -0.006
-0.001  0.006   0.004   0.001   0.039   0.011   0.  0.  -0.032  0.007
0.027   0.058   0.02    0.009   0.109   0.026   0.  0.  -0. -0.002]
max значение признака после масштабирования:
[ 0.958 0.815   0.956   0.894   0.811   1.22    0.88        0.933   0.932   1.0
37
0.427   0.498   0.441   0.284   0.487   0.739   0.767   0.629   1.337   0.391
0.896   0.793   0.849   0.745   0.915   1.132   1.07        0.924   1.205   1.6
31]
```

Возможно, полученные результаты несколько удивят вас: после масштабирования минимальные и максимальные значения признаков в тестовом наборе не равны 0 и 1. Некоторые признаки даже выходят за пределами диапазона 0-1! Объяснить это можно тем, что **MinMaxScaler** (и все остальные типы масштабирования) всегда применяют одинаковое преобразование к обучающему и тестовому наборам. Это означает, что метод transform всегда вычитает минимальное значение, вычисленное для обучающего набора, и делит на ширину диапазона, вычисленную также для обучающего набора. Минимальное значение и ширина диапазона для обучающего набора могут отличаться от минимального значения и ширины диапазона для тестового набора.

Масштабирование обучающего и тестового наборов одинаковым образом

Чтобы модель контролируемого обучения работала на тестовом наборе, важно преобразовать обучающий и тестовый наборы одинаковым образом. Пример (рис. 2) показывает, что произошло бы, если бы мы использовали минимальное значение и ширину диапазона, отдельно вычисленные для тестового набора:

```
from sklearn.datasets import make_blobs
X, _ = make_blobs(
    n_samples=50,
    centers=5,
    random_state=4,
    cluster_std=2
)
X_train, X_test = train_test_split(X, random_state=5, test_size=.1)
fig, axes = plt.subplots(1, 3, figsize=(13, 4))
axes[0].scatter(X_train[:, 0], X_train[:, 1],
    c=mglearn.cm2(0), label="Обучающий набор", s=60)
axes[0].scatter(X_test[:, 0], X_test[:, 1], marker='^',
    c=mglearn.cm2(1), label="Тестовый набор", s=60)
axes[0].legend(loc='upper left')
axes[0].set_title("Исходные данные")

scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

axes[1].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1],
    c=mglearn.cm2(0), label="Обучающий набор", s=60)

axes[1].scatter(
    X_test_scaled[:, 0],
    X_test_scaled[:, 1],
    marker='^',
    c=mglearn.cm2(1),
    label="Тестовый набор",
    s=60
)
axes[1].set_title("Масштабированные данные")

test_scaler = MinMaxScaler()
test_scaler.fit(X_test)
X_test_scaled_badly = test_scaler.transform(X_test)

axes[2].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1],
```

```
c=mglearn.cm2(0), label="Обучающий набор", s=60)

axes[2].scatter(
    X_test_scaled_badly[:, 0],
    X_test_scaled_badly[:, 1],
    marker='^',
    c=mglearn.cm2(1),
    label="Тестовый набор",
    s=60
)
axes[2].set_title("Неправильно масштабированные данные")
for ax in axes:
    ax.set_xlabel("Признак 0")
    ax.set_ylabel("Признак 1")
plt.show()
```

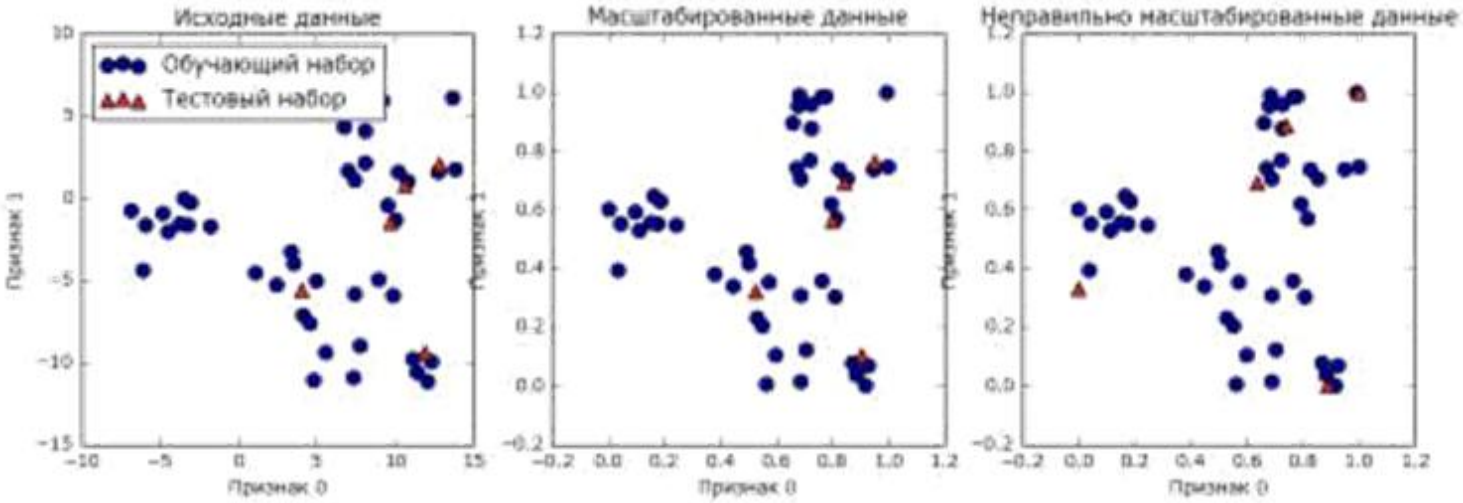

Рис. 2 Результаты одинакового масштабирования обучающего и тестового наборов (центр) и отдельного масштабирования обучающего и тестового наборов (справа)

Первый график – это немасштабированный двумерный массив данных, наблюдения обучающего набора показаны кружками, а наблюдения тестового набора показаны треугольниками. Второй график – те же самые данные, но масштабированы с помощью **MinMaxScaler**. Здесь мы вызвали метод для обучающего набора, а затем вызвали метод transform для обучающего и тестового наборов. Как видите, набор данных на втором графике идентичен набору, приведенному на первом графике, изменились лишь метки осей. Теперь все признаки принимают значения в диапазоне от 0 до 1. Кроме того, видно, что минимальные и максимальные значения признаков в тестовом наборе (треугольники) не равны 0 и 1.

Третий график показывает, что произойдет, если отмасштабируем обучающий и тестовый наборы по отдельности. В этом случае минимальные и максимальные значения признаков в обучающем и тестовом наборах равны 0 и 1. Но теперь набор данных выглядит иначе. Тестовые точки причудливым образом сместились, поскольку масштабированы по-другому. Мы изменили расположение данных произвольным образом. Очевидно, это совсем не то. что нам нужно.

Еще один способ задуматься о неправильности этих действий заключается в том, что представить тестовый набор в виде одной точки. Не существует способа правильно масштабировать единственную точку данных, чтобы с помощью **MinMaxScaler** получить значения минимума и максимума. Однако размер тестового набора не должен влиять на обработку данных.

Быстрые и эффективные альтернативные способы подгонки моделей

Как правило, вам нужно сначала подогнать модель на некотором наборе данных с помощью метода **fit**, а затем выполнить преобразование набора с помощью метода **transform**. Это весьма распространенная задача, которую можно выполнить более эффективно, чем просто вызвать метод **fit**, а затем вызвать метод **transform**. Что касается вышеописанного случая, все модели, которые используют метод **transform**, также позволяют воспользоваться методом **fit_transform**. Ниже дан пример использования **StandardScaler**:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit(X).transform(X)
X_scaled_d = scaler.fit_transform(X)
```

Несмотря на то что применение **fit_transform** вовсе не обязательно будет более эффективным для всех моделей, эффективная практика заключается в использовании этого метода для преобразования обучающего набора.

Влияние предварительное обработки на машинное обучение с учителем

Теперь давайте вернемся к набору данных **cancer** и посмотрим, как использование **MinMaxScaler** повлияет на обучение **SVC** (мы выполняем то же самое масштабирование, что делали в главе 2, но другим способом). Во-первых, давайте для сравнения снова подгоним **SVC** на исходных данных:

```
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data,
    cancer.target,
    random_state=0
)
svm = SVC(C=100)
svm.fit(X_train, y_train)
print("Правильность на тестовом наборе: {:.2f}"
    .format(svm.score(X_test, y_test))
)
```

```
Правильность на тестовом наборе: 0.63
```

Теперь давайте отмасштабируем данные с помощью **MinMaxScaler** перед тем, как подгонять **SVC**:

```
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm.fit(X_train_scaled, y_train)
print("Правильность на масштабированном тестовом наборе: {:.2f}"
    .format( svm.score(X_test_scaled, y_test))
)
```

```
Правильность на масштабированном тестовом наборе: 0.97
```

Как мы уже видели ранее, эффект масштабирования данных весьма существенен. Хотя масштабирование данных не предполагает каких-либо сложных математических расчетов, эффективная практика заключается в том, чтобы использовать методы масштабирования, предлагаемые scikit-learn, а не создавать их заново самостоятельно, поскольку легко ошибиться даже в этих простых вычислениях.

Кроме того, можно легко заменить один алгоритм предварительной обработки на другой, сменив имя используемого класса, поскольку все классы предварительной обработки имеют один и тот же интерфейс, состоящий из методов **fit** и **transform**:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm.fit(X_train_scaled, y_train)
print("Правильность SVM на тестовом наборе: {:.2f}"
    .format(svm.score(X_test_scaled, y_test))
)
```

```
Правильность SVM на тестовом наборе: 0.96
```

Теперь узнав, как работают простые преобразования, выполняющие предварительную обработку данных, давайте перейдем к более интересным преобразованиям, использующим машинное обучение без учителя.

Снижение размерности, выделение признаков и множественное обучение

Как мы уже говорили ранее, преобразование данных с помощью неконтролируемого обучения может быть обусловлено многими причинами. Наиболее распространенные причины – визуализация, сжатие данных, а также поиск такого представления данных, которое даст больше информации в ходе дальнейшей обработки.

Одним из самых простых и наиболее широко используемых алгоритмов контролируемого обучения является анализ главных компонент (**principal component analysis**, **PCA**). Кроме того, мы рассмотрим еще два алгоритма: факторизацию неотрицательных матриц (**non-negative matrix factorization**, **NMF**), которая обычно используется для выделения признаков, и стохастическое вложение соседей с распределением Стьюдента (**t-distributed stochastic neighbor embedding**, **t-SNE**), которое обычно используется для визуализации с использованием двумерных диаграмм рассеяния.

Код к лабораторной работе:

```
import mglearn
import sklearn
import matplotlib.pyplot as plt


"In[2]:"
mglearn.plots.plot_scaling()
plt.show()


"In[3]:"
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=1)
print(X_train.shape)
print(X_test.shape)


"In[4]:"
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()


"In[5]:"
scaler.fit(X_train)


"In[6]:"
min_on_training = X_train.min(axis=0)
range_on_training = (X_train - min_on_training).max(axis=0)
X_train_scaled = (X_train - min_on_training) / range_on_training
print("форма преобразованного массива: {}".format(X_train_scaled.shape))
print("min значение признака до масштабирования:\n {}".format(X_train.min(axis=0)))
print("max значение признака до масштабирования:\n {}".format(X_train.max(axis=0)))
print("min значение признака после масштабирования:\n {}".format( X_train_scaled.min(axis=0)))
print("max значение признака после масштабирования:\n {}".format( X_train_scaled.max(axis=0)))


"In[7]:"
X_test_scaled = (X_test - min_on_training) / range_on_training
print("min значение признака после масштабирования:\n{}".format(X_test_scaled.min(axis=0)))
print("max значение признака после масштабирования:\n{}".format(X_test_scaled.max(axis=0)))


"In[8]:"
from sklearn.datasets import make_blobs
X, _ = make_blobs(n_samples=50, centers=5, random_state=4, cluster_std=2)
```

```
X_train, X_test = train_test_split(X, random_state=5, test_size=.1)
fig, axes = plt.subplots(1, 3, figsize=(13, 4))
axes[0].scatter(X_train[:, 0], X_train[:, 1],
    c=mglearn.cm2(0), label="Обучающий набор", s=60)
axes[0].scatter(X_test[:, 0], X_test[:, 1], marker='^',
    c=mglearn.cm2(1), label="Тестовый набор", s=60)
axes[0].legend(loc='upper left')
axes[0].set_title("Исходные данные")

scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

axes[1].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1],
    c=mglearn.cm2(0), label="Обучающий набор", s=60)

axes[1].scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], marker='^', c=mglearn.cm2(1), label="Тестовый набор", s=60)
axes[1].set_title("Масштабированные данные")

test_scaler = MinMaxScaler()
test_scaler.fit(X_test)
X_test_scaled_badly = test_scaler.transform(X_test)

axes[2].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1],
c=mglearn.cm2(0), label="Обучающий набор", s=60)

axes[2].scatter(X_test_scaled_badly[:, 0], X_test_scaled_badly[:, 1], marker='^', c=mglearn.cm2(1), label="Тестовый на
бор", s=60)
axes[2].set_title("Неправильно масштабированные данные")
for ax in axes:
    ax.set_xlabel("Признак 0")
    ax.set_ylabel("Признак 1")
plt.show()


"In[9]:"
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit(X).transform(X)
X_scaled_d = scaler.fit_transform(X)


"In[10]:"
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=0)
svm = SVC(C=100)
svm.fit(X_train, y_train)
```

```
print("Правильность на тестовом наборе: {:.2f}".format(svm.score(X_test, y_test)))



"In[11]:"
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm.fit(X_train_scaled, y_train)
print("Правильность на масштабированном тестовом наборе: {:.2f}".format( svm.score(X_test_scaled, y_test)))



"In[12]:"
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
svm.fit(X_train_scaled, y_train)
print("Правильность SVM на тестовом наборе: {:.2f}".format(svm.score(X_test_scaled, y_test)))
```