Лабораторная работа №30

Анализ главных компонент (PSA)

Анализ главных компонент представляет собой метод, который осуществляет вращение данных с тем, чтобы преобразованные признаки не коррелировали между собой. Часто это вращение сопровождается выбором подмножества новых признаков в зависимости от их важности с точки зрения интерпретации данных. Следующий пример (рис.1) иллюстрирует результат применения **PCA** к синтетическому двумерному массиву данных:

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

Первый рис. (вверху слева) показывает исходные точки данных, выделенные цветом для лучшей дискриминации. Алгоритм начинает работу с того, что сначала находит направление максимальной дисперсии, помеченное как «компонента 1». Речь идет о направлении (или векторе) данных, который содержит большую часть информации, или другими словами, направление, вдоль которого признаки коррелируют друг с другом сильнее всего. Затем алгоритм находит направление, которое содержит наибольшее количество информации, и при этом ортогонально (расположено под прямым углом) первому направлению. В двумерном пространстве существует только одна возможная ориентация, расположенная под прямым углом, но в пространствах большей размерности может быть (бесконечно) много ортогональных направлений. Хотя эти две компоненты изображаются в виде стрелок, на самом деле не имеет значения, где начало, а где конец, мы могли бы нарисовать первую компоненту, выходящую из центра в верхний левый угол, а не в нижний правый. Направления, найденные с помощью этого алгоритма, называются главными компонентами (**principal components**), поскольку они являются основными направлениями дисперсии данных. В целом максимально возможное количество главных компонент равно количеству исходных признаков.

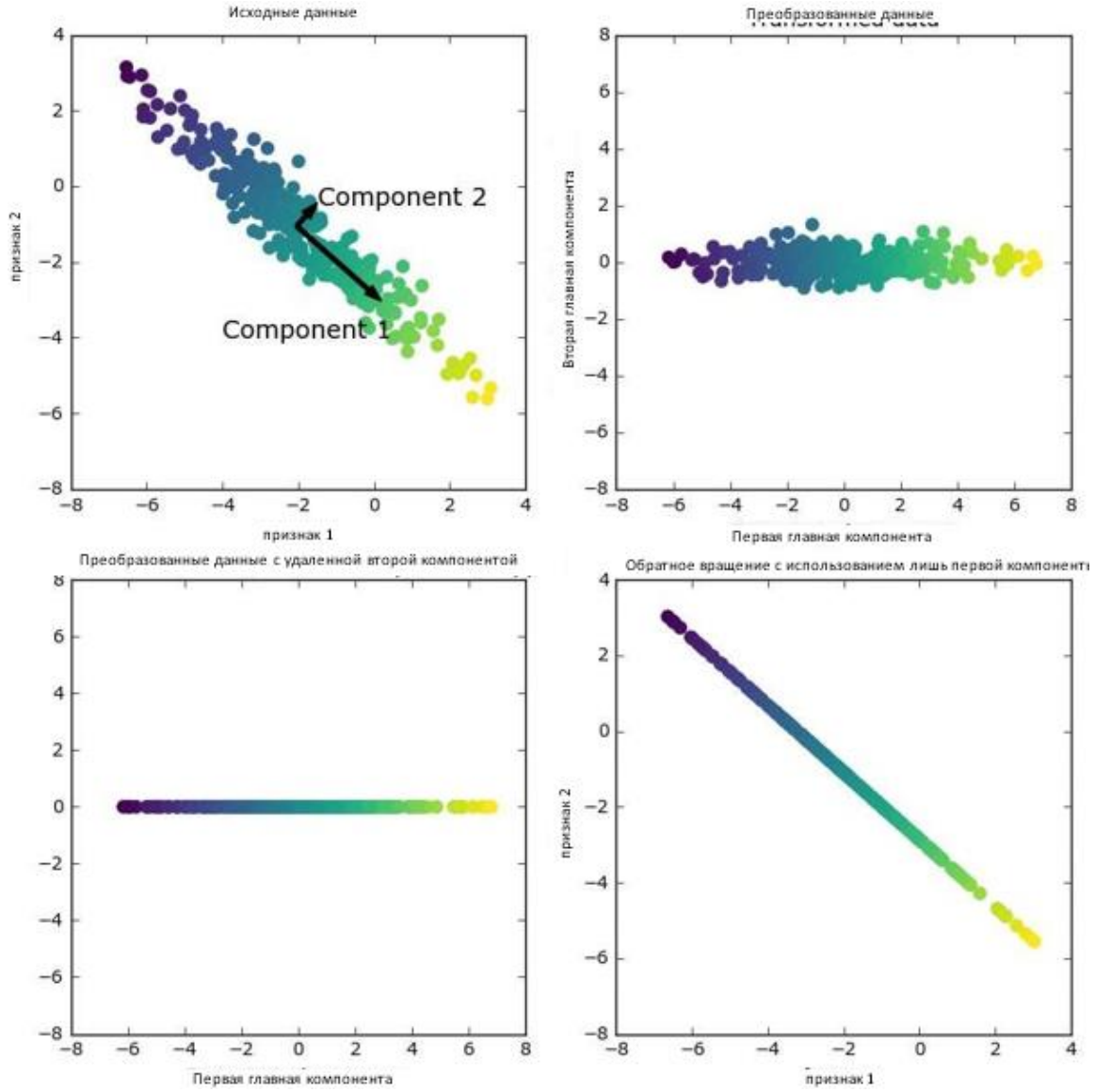

Рис. 1 Преобразование данных с помощью PCA

Второй график (вверху справа) показывает те же самые данные, но теперь повернутые таким образом, что первая главная компонента совпадает с осью x, а вторая главная компонента совпадает с осью y. Перед вращением из каждого значения данных вычитается среднее, таким образом, преобразованные данные центрированы около нуля. В новом представлении данных, найденном с помощью **PCA**, две оси становятся некоррелированными. Это означает, что в новом представлении все элементы корреляционной матрицы данных, кроме диагональных, будут равны нулю.

Мы можем использовать **PCA** для уменьшения размерности, сохранив лишь несколько главных компонент. В данном примере мы можем оставить лишь первую главную компоненту, как показано на третьем графике рис.1 (внизу слева). Это уменьшит размерность данных: из двумерного массива данных получаем одномерный массив данных. Однако следует отметить, что вместо того, чтобы оставить лишь один из исходных признаков, мы находим наиболее интересное направление (выходящее из верхнего левого угла в нижний правый на первом графике) и оставляем это направление, т.е. первую главную компоненту. И, наконец, мы можем отменить вращение и добавить обратно среднее значение к значениям данных. В итоге получим данные, показанные на

последнем графике рис.1. Эти точки располагаются в пространстве исходных признаков, но мы оставили лишь информацию, содержащуюся в первой главной компоненте. Это преобразование иногда используется, чтобы удалить эффект шума из данных или показать, какая часть информации сохраняется при использовании главных компонент.

Применение PCA к набору данных cancer для визуализации

Одним из наиболее распространенных применений **PCA** является визуализация высокоразмерных наборов данных, довольно сложно построить диаграммы рассеяния для данных, которые включают больше двух признаков. Для набора данных Iris мы смогли построить матрицу диаграмм рассеяния, которая дала нам частичное представление о данных, показав все возможные комбинации двух признаков. Но если мы захотим взглянуть на набор данных Breast Cancer, использование матрицы диаграмм рассеяния будет затруднительным. Этот набор данных содержит 30 признаков, которые привели бы к 30 * 14 = 420 диаграммам рассеяния! Мы никогда не сможем детально просмотреть все эти графики, не говоря уже об их интерпретации.

Впрочем, можно воспользовать более простой визуализацией, вычислив гистограммы распределения значений признаков для двух классов, доброкачественных и злокачественных опухолей (рис. 2):

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

fig, axes = plt.subplots(15, 2, figsize=(10, 20))
malignant = cancer.data[cancer.target == 0]
benign = cancer.data[cancer.target == 1]
ax = axes.ravel()
for i in range(30):
    _, bins = np.histogram(cancer.data[:, i], bins=50)
    ax[i].hist(malignant[:, i], bins=bins, color=mglearn.cm3(0), alpha=.5)
    ax[i].hist(benign[:, i], bins=bins, color=mglearn.cm3(2), alpha=.5)
    ax[i].set_title(cancer.feature_names[i])
    ax[i].set_yticks(())
    ax[0].set_xlabel("Значение признака")
    ax[0].set_ylabel("Частота")
ax[0].legend(["доброкачественная", "злокачественная"], loc="best")
fig.tight_layout()
plt.show()
```

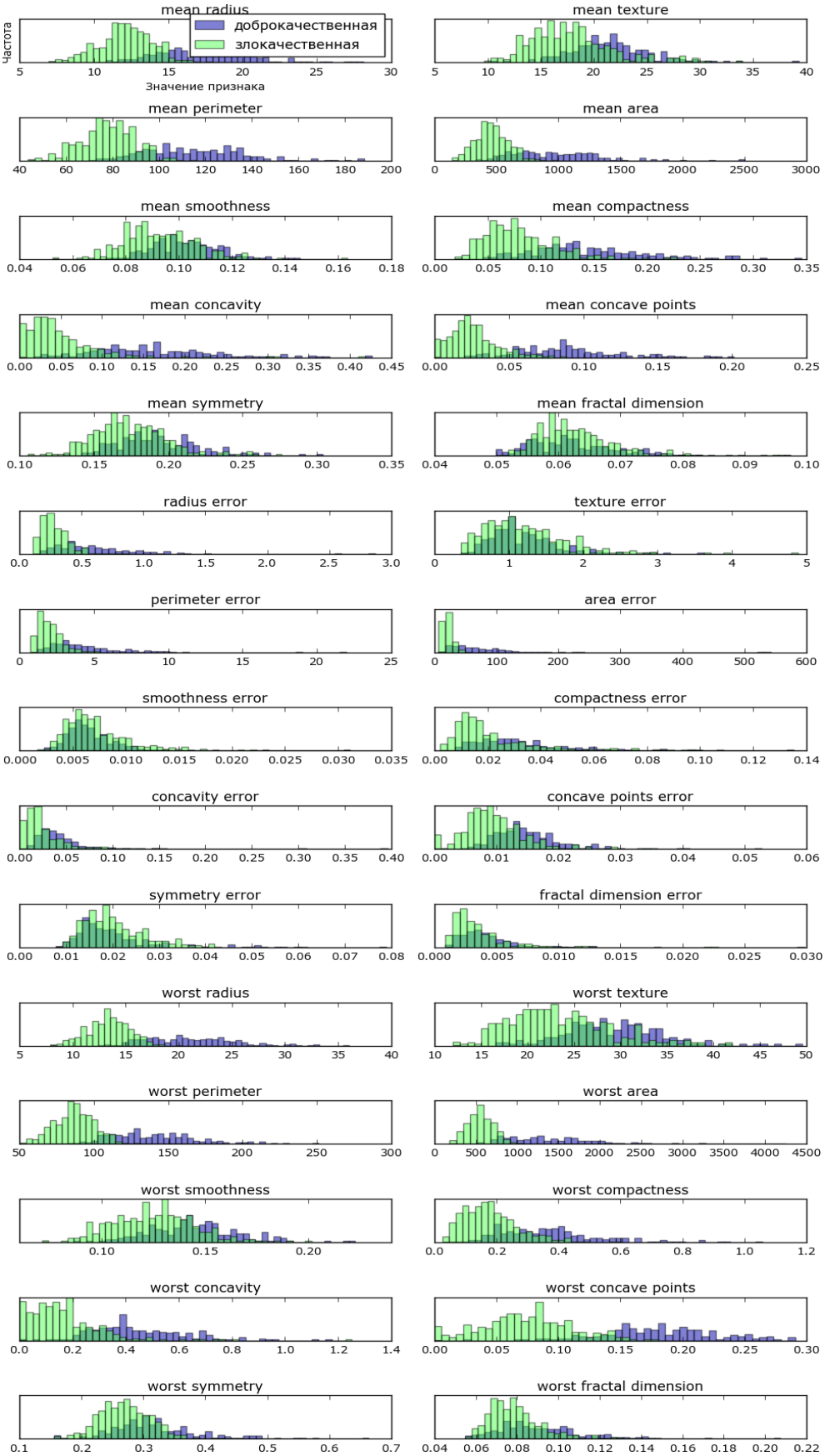

Рис. 2 Преобразование данных с помощью PCA

В данном случае мы строим для каждого признака гистограмму, подсчитывая частоту встречаемости точек данных в пределах границ интервалов (этот интервал еще называют бином). Каждый график содержит две наложенные друг на друга гистограммы, первая – для всех точек, относящихся к классу «доброкачественная опухоль» (синий цвет), а вторая – для всех точек, относящихся к классу «злокачественная опухоль» (зеленый цвет). Это дает нам некоторое представление о распределении каждого признака по двум классам и позволяет нам строить предположения о том, какие признаки лучше всего дискриминируют злокачественные и доброкачественные опухоли. Например, признак «smoothness error», похоже, довольно малоинформативен, потому что две гистограммы, построенные для данного признака, большей частью накладываются друг на друга, в то время признак «worst concave points» кажется весьма информативным, поскольку гистрограммы, построенные для этого признака, практически не накрывают друг друга.

Однако этот график не дает нам никакой информации о взаимодействии между переменными и взаимосвязях между признаками и классами зависимой переменной. Используя PCA, мы можем учесть главные взаимодействия и получить несколько более полную картину. Мы можем найти первые две главные компоненты и визуализировать данные в этом новом двумерном пространстве с помощью одной диаграммы рассеяния.

Перед тем, как применить **PCA**, мы отмасштабируем наши данные таким образом, чтобы каждый признак имел единичную дисперсию, воспользовавшись **StandardScaler**:

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler

cancer = load_breast_cancer()
scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
```

Обучение **PCA** и его применение так же просто, как применение преобразований, выполняющихся в ходе предварительной обработки. Мы создаем экземпляр объекта **PCA**, находим главные компоненты, вызвав метод **fit**, а затем применяем вращение и снижение размерности, вызвав метод **transform**. По умолчанию **PCA** лишь поворачивает (и смещает) данные, но сохраняет все главные компоненты. Чтобы уменьшить размерность данных, нам нужно указать, сколько компонент мы хотим сохранить при создании объекта **PCA**:

```
from sklearn.decomposition import PCA
# оставляем первые две главные компоненты
pca = PCA(n_components=2)
#  подгоняем  модель  PCA  на  наборе  данных  breast  cancer
pca.fit(X_scaled)
# преобразуем данные к первым двум главным компонентам
X_pca = pca.transform(X_scaled)
print("Форма исходного массива: {}"
    .format(str(X_scaled.shape))
)
print("Форма массива после сокращения размерности: {}"
    .format(str(X_pca.shape))
)
```

Теперь мы можем построить график первых двух главных компонент

```
# строим график первых двух главных компонент, классы выделены цветом
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], cancer.target)
plt.legend(cancer.target_names, loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("Первая главная компонента")
plt.ylabel("Вторая главная компонента")
plt.show()
```

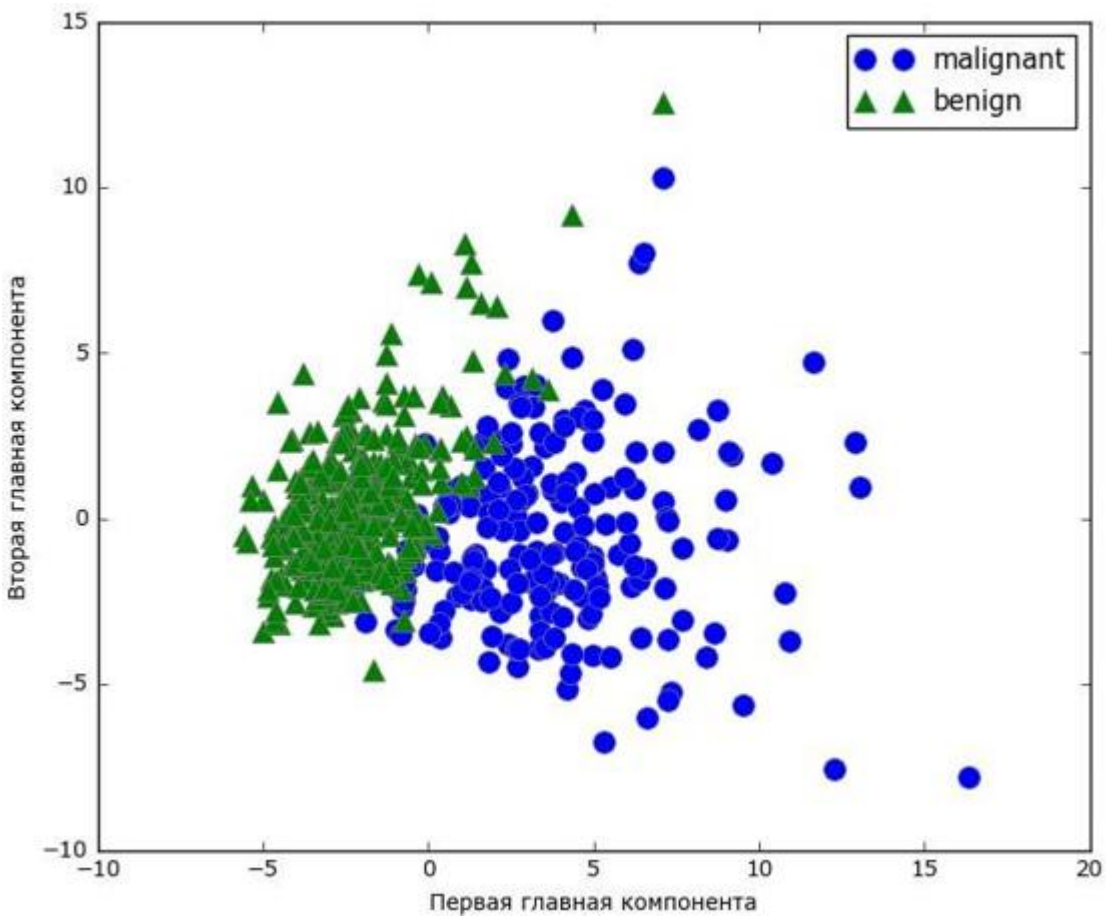

Рис. 3 Двумерная диаграмма рассеяния для набора данных Breast Cancer с использованием первых двух главных компонент

Важно отметить, что **PCA** является методом машинного обучения без учителя и не использует какой-либо информации о классах при поиске поворота. Он просто анализирует корреляционные связи в данных. Для точечного графика, показанного здесь, мы построили график, где по оси x отложена первая главная компонента, по оси y – вторая главная компонента, а затем воспользовались информацией о классах, чтобы выделить точки разным цветом. Вы можете увидеть, что в рассматриваемом двумерном пространстве эти два класса разделены достаточно хорошо. Это наводит на мысль, что даже линейный классификатор (который проведет прямую линию в этом пространстве) сможет достаточно хорошо разделить два класса. Кроме того, мы можем увидеть, что случаи злокачественных опухолей (синие точки) более распространены, чем случаи

доброкачественных опухолей (зеленые точки) – что отчасти было видно на гистограммах рис. 2.

Недостаток **PCA** заключается в том, что эти две оси графика часто бывает сложно интерпретировать. Главные компоненты соответствуют направлениям данных, поэтому они представляют собой комбинации исходных признаков. Однако, как мы скоро увидим, эти комбинации обычно очень сложны. Сами главные компоненты могут быть сохранены в атрибуте **components_** объекта **PCA** в ходе подгонки:

```
print("форма главных компонент: {}".format(pca.components_.shape))
```

Каждая строка в атрибуте **components_** соответствует одной главной компоненте и они отсортированы по важности (первой приводится первая главная компонента и т.д.). Столбцы соответствуют атрибуту исходных признаков для объекта **PCA** в этом примере, «mean radius», «mean texture» и т.д. Давайте посмотрим на содержимое атрибута **components_**:

```
print("компоненты PCA:\n{}".format(pca.components_))
```

Кроме того, с помощью тепловой карты (рис. 4) мы можем визуализировать коэффициенты, чтобы упростить их интерпретацию:

```
plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1], ["Первая компонента", "Вторая компонента"])
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)), cancer.feature_names, rotation=60,
 ha='left')
plt.xlabel("Характеристика")
plt.ylabel("Главные компоненты")
plt.show()
```

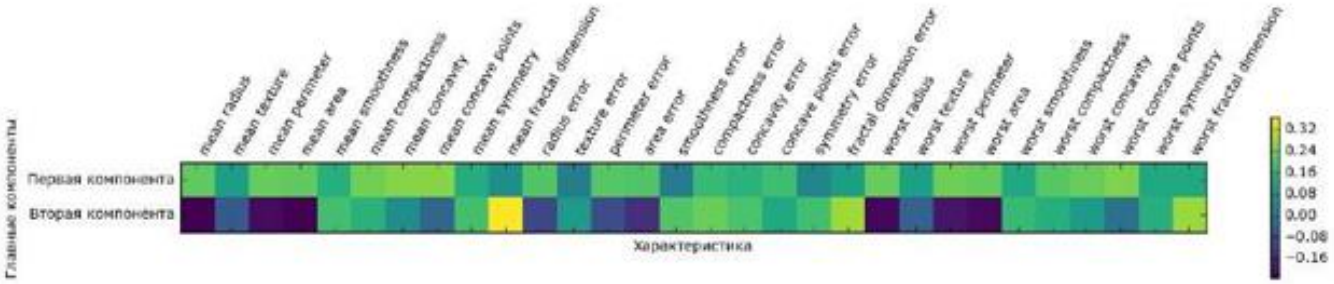

Рис. 4 Тепловая карта первых двух главных компонент для набора данных рака Breast Cancer

Вы можете увидеть, что в первой компоненте коэффициенты всех признаков имеют одинаковый знак (они положительные, но, как мы уже говорили ранее, не имеет значения, какое направление указывает стрелка). Это означает, что существует общая корреляция между всеми признаками. Высоким значениям одного признака будут соответствовать высокие значения остальных признаков. Во второй компоненте коэффициенты признаков имеют разные знаки. Обе компоненты включают все 30 признаков. Смешивание всех признаков – это как раз то, что усложняет интерпретацию осей на рис. 4.

Метод «Собственых лиц» (eigenfaces) для выделения характеристик

Еще одно применение **PCA**, о котором мы уже упоминали ранее, – это выделение признаков. Идея, лежащая в основе выделения признаков, заключается в поиске нового представления данных, которое в отличие от исходного лучше подходит для анализа. Отличный пример, показывающий, что выделение признаков может быть полезно, – это работа с изображениями. Изображения состоят из пикселей, обычно хранящихся в виде интенсивностей красной, зеленой и синей составляющих цвета (RGB). Объекты в изображениях, как правило, состоят из тысяч пикселей и лишь все вместе эти пиксели приобретают смысл.

Мы приведем очень простой пример того, как можно применить выделение признаков к изображениям с помощью **PCA**. Для этого мы воспользуемся набором данных Labeled Faces in the Wild. Этот набор данных содержит изображения лиц знаменитостей, загруженных из Интернета, и включает в себя лица политиков, певцов, актеров и спортсменов с начала 2000-х годов. Мы преобразуем эти фотографии в оттенки серого, а также уменьшим их для более быстрой обработки. Вы можете увидеть некоторые изображения на рис. 5:

```
# conda install -c anaconda openssl
from sklearn.datasets import fetch_lfw_people

people = fetch_lfw_people(min_faces_per_person=20, resize=0.7)
image_shape = people.images[0].shape
fix, axes = plt.subplots(
    2,
    5,
    figsize=(15, 8),
    subplot_kw={
        'xticks': (),
        'yticks': ()
    }
)
for target, image, ax in zip(
    people.target,
    people.images,
    axes.ravel()
):
    ax.imshow(image)
    ax.set_title(people.target_names[target])
plt.show()
```

Рис. 5 Некоторые изображения из набора данных Labeled Faces in the Wild

Получаем 3023 изображения размером 87 x 65 пикселей, принадлежащие 62 различным людям:

```
print("форма массива изображений лиц: {}"
    .format(people.images.shape)
)
print("количество классов: {}"
    .format(len(people.target_names))
)
```

Однако данные немного асимметричны. Как вы можете здесь увидеть, он содержит большое количество изображений Джорджа Буша и Колина Пауэлла:

```
# вычисляем частоту встречаемости каждого ответа
counts = np.bincount(people.target)
# печатаем частоты рядом с ответами
for i, (count, name) in enumerate(zip(counts, people.target_names)):
    print("{0:25} {1:3}".format(name, count), end=' ')
    if (i + 1) % 3 == 0:
        print()
```

Чтобы данные стали менее асимметричными, мы будем рассматривать не более 50 изображений каждого человека (в противном случае выделение признаков будет перегружено большим количеством изображений Джорджа Буша):

```
mask = np.zeros(people.target.shape, dtype=np.bool)
for target in np.unique(people.target):
    mask[np.where(people.target == target)[0][:50]] = 1
X_people = people.data[mask]
y_people = people.target[mask]
#  для  получения  большей  стабильности
# масштабируем  шкалу  оттенков  серого  так,  чтобы  значения
# были в диапазоне от 0 до 1
# вместо использования шкалы значений от 0 до 255
X_people = X_people / 255.
```

Общая задача распознавания лиц заключается в том, чтобы спросить, не принадлежит ли незнакомое фото уже известному человеку из базы данных. Она применяется при составлении фотоколлекций, в социальных сетях и программах обеспечения безопасности. Один из способов решения этой задачи заключается в построении классификатора, в котором каждый человек представляет собой отдельный класс. Однако изображения, записанные в базах лиц, обычно принадлежат большому количеству самых различных людей и при этом очень мало фотографий принадлежат одному и тому же человеку (то есть очень мало обучающих примеров, принадлежащих одному классу). Для большинства классификаторов это представляет проблему. Кроме того, часто необходимо добавить фотографии новых людей, при этом не перестраивая заново огромную модель.

Самое простое решение – использовать классификатор одного ближайшего соседа, который ищет лицо, наиболее схожее с классифицируемым. Этот классификатор в принципе может работать только с одним обучающим примером в классе. Давайте посмотрим, насколько хорошо здесь сработает **KNeighborsClassifier**:

```
from sklearn.neighbors import KNeighborsClassifier

# разбиваем данные на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(
    X_people, y_people, stratify=y_people, random_state=0)
#  строим  KNeighborsClassifier  с  одним  соседом
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("Правильность на тестовом наборе для 1-nn: {:.2f}"
    .format(knn.score(X_test, y_test))
)
```

Мы получаем правильность 26.6%, на самом деле это неплохо для классификационной задачи с 62 классами (случайное угадывание даст вам правильность около 1/62 = 1.61%), но и не так велико. Мы правильно распознаем лишь каждое четвертое изображение человека.

И вот именно здесь применяется **PCA**. Вычисление расстояний в исходном пиксельном пространстве – довольно неудачный способ измерить сходство между лицами. Используя пиксельное представление для сопоставления двух изображений, мы сравниваем значение каждого отдельного пикселя по шкале градаций серого со значением пикселя в соответствующем положении на другом изображении. Это представление довольно сильно отличается от интерпретации изображений лиц людьми и крайне трудно выделить характеристики лица с использованием этого исходного

представления. Например, использование пиксельных расстояний означает, что смещение лица на один пиксель вправо соответствует резкому изменению, дающему совершенно другое представление данных. Мы рассчитываем на то, что использование расстояний вдоль главных компонент может улучшить правильность. Здесь мы воспользуемся опцией **PCA** выбеливание (**whitening**), которая преобразует компоненты к одному и тому же масштабу. Операция выбеливания аналогична применению **StandardScaler** после преобразования. Повторно используя данные, приведенные на рис. 1, выбеливание не только поворачивает данные, но и масштабирует их таким образом, чтобы центральный график представлял собой окружность вместо эллипса (см. рис. 6):

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

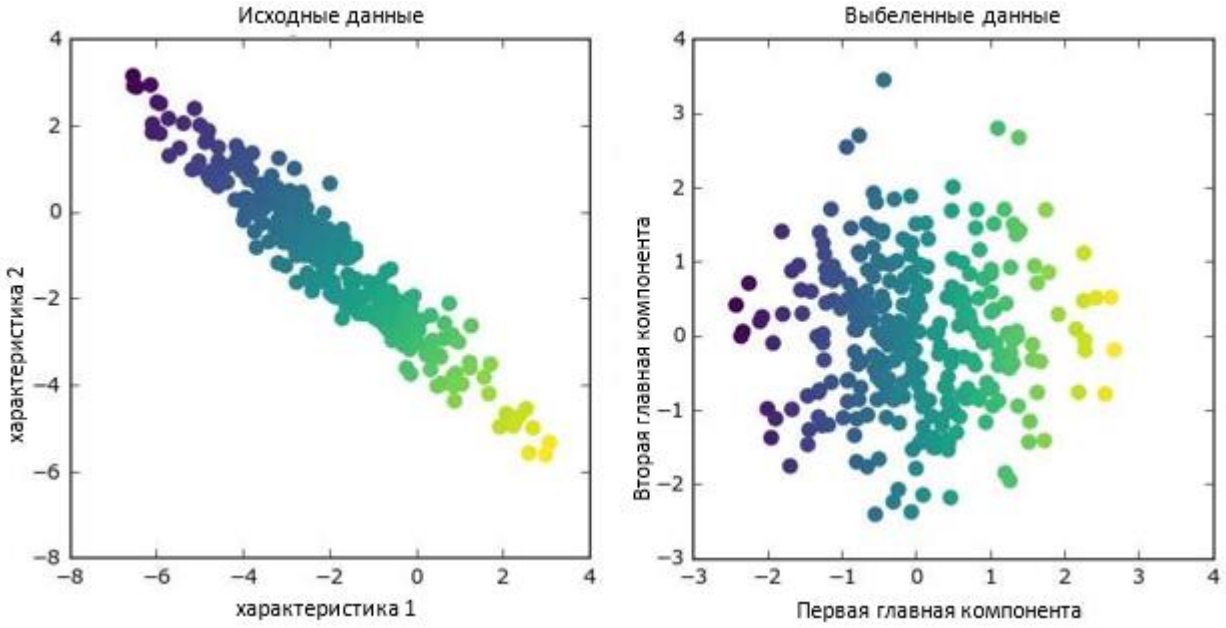

Рис. 6 Преобразование данных с использованием выбеливания

Мы подгоняем объект **PCA** на обучающих данных и извлекаем первые 100 главных компонент. Затем мы преобразуем обучающие и тестовые данные:

```
pca = PCA(n_components=100, whiten=True, random_state=0).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("обучающие данные после PCA: {}".format(X_train_pca.shape))
```

Новые данные содержат 100 новых признаков, первые 100 главных компонент. Теперь мы можем использовать новое представление, чтобы классифицировать наши изображения, используя классификатор одного ближайшего соседа:

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
print("Правильность на тестовом наборе: {:.2f}"
    .format(knn.score(X_test_pca, y_test))
)
```

Наша правильность улучшилась весьма значительно, с 26.6% до 35.7%, это подтверждает наше предположение о том, что главные компоненты могут дать лучшее представление данных.

Работая с изображениями, мы можем легко визуализировать найденные главные компоненты. Вспомним, что компоненты соответствуют направлениям в пространстве входных данных. Пространство входных данных здесь представляет собой изображения в градациях серого размером 87x65 пикселей, поэтому направления внутри этого пространства также являются изображениями в градациях серого размером 87x65 пикселей.

Давайте посмотрим на первые несколько главных компонент (рис. 7):

```
print("форма pca.components_: {}".format(pca.components_.shape))
```

Несмотря на то что мы, конечно, не сможем понять весь содержательный смысл этих компонент, мы можем догадаться, какие характеристики изображений лиц были выделены некоторыми компонентами. Похоже, что первая компонента главным образом кодирует контраст между лицом и фоном, а вторая компонента кодирует различия в освещенности между правой и левой половинами лица и т.д. Хотя это представление данных в отличие от исходных значений пикселей немного содержательнее, оно по-прежнему весьма далеко от того, как человек привык воспринимать лицо. Поскольку модель **PCA** основана на пикселях, выравнивание изображения лица (положения глаз, подбородка и носа) и освещенность оказывают сильное влияние на степень сходства двух пиксельных изображений. Однако выравнивание и освещенность, вероятно, будут совсем не теми характеристиками, которые человек будет воспринимать в первую очередь. Когда людей просят оценить сходство между лицами, они в большей степени руководствуются такими признаками, как возраст, пол, выражение лица и прическа, то есть признаками, которые трудно выделить, исходя из интенсивностей пикселей. Важно помнить, что, как правило, алгоритмы в отличие от человека интерпретируют данные (в частности, визуальные данные, например, изображения популярных людей) совершенно по- другому.

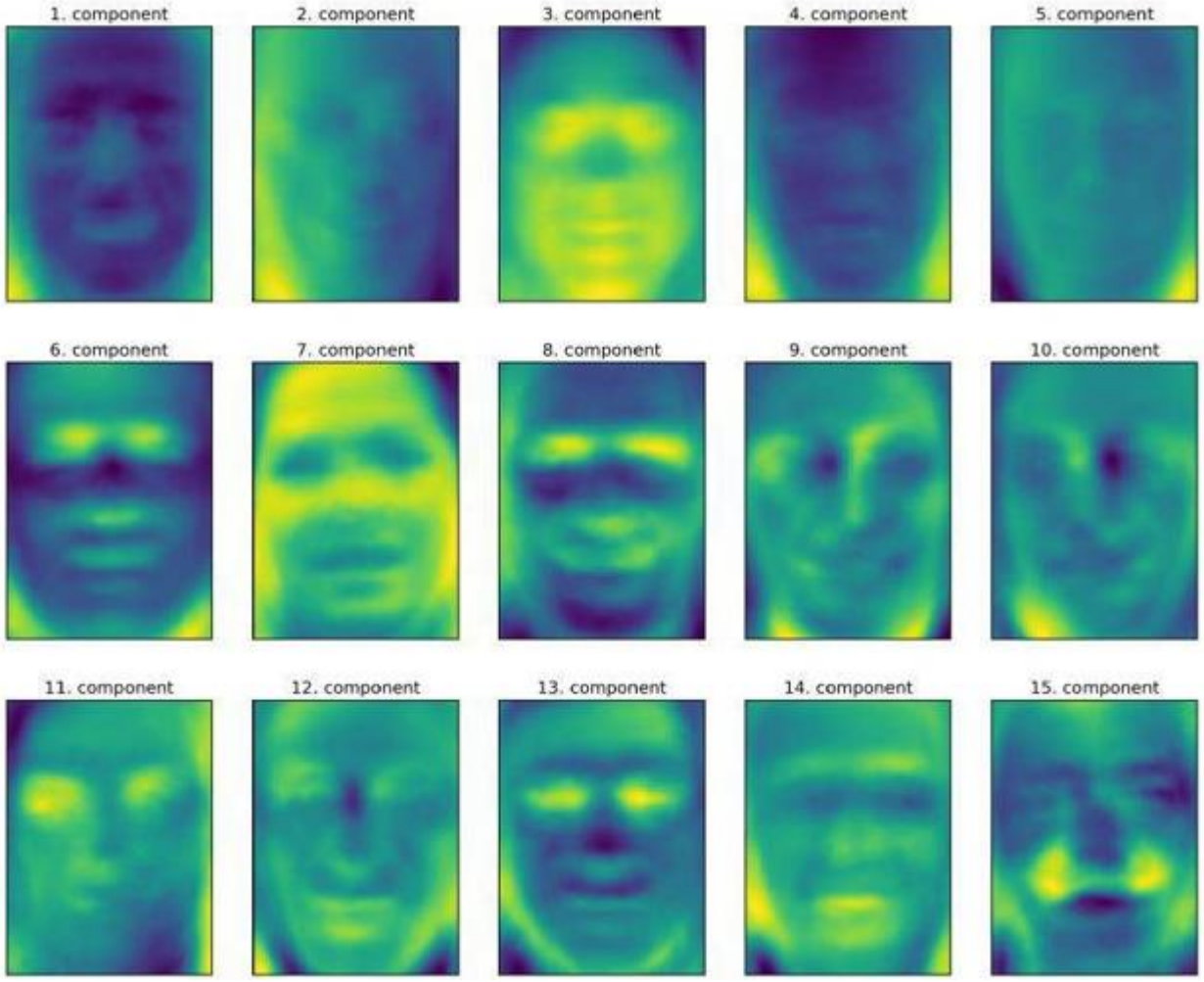

Рис. 7 Собственные векторы первых 15 компонент для набора лиц

Впрочем, давайте вернемся к конкретному случаю использования **PCA**. Мы кратко рассказали о преобразовании **PCA** как способе поворота данных с последующим удалением компонент, имеющих низкую дисперсию. Еще одна полезная интерпретация заключается в том, чтобы попытаться вычислить значения новых признаков, полученные после поворота **PCA**, таким образом, мы можем записать тестовые точки в виде взвешенной суммы главных компонент (см. рис. 8).

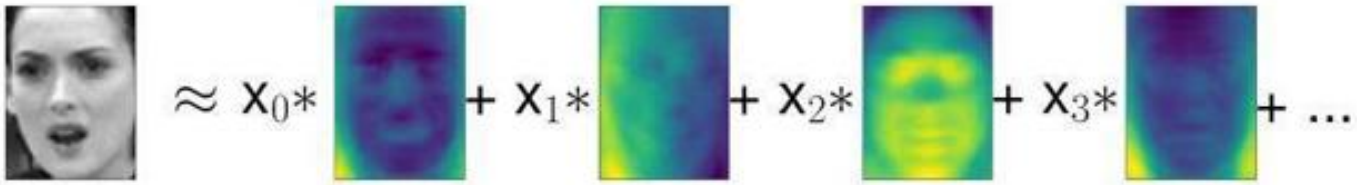

Рис. 8 Схематическое изображение PCA, осуществляющего разложение изображения на взвешенную сумму компонент

Здесь x , x и т. д. являются коэффициентами главных компонент для конкретной точки данных, другими словами, они представляют собой изображение в новом пространстве, полученном в результате вращения. Еще один способ понять, что делает

модель **PCA** – реконструировать исходные данные, используя лишь некоторые компоненты. На третьем графике рис.1 мы удалили вторую компоненту, затем мы отменили вращение и добавили обратно среднее значение, чтобы получить новые точки в исходном пространстве с удаленной второй компонентой, как показано на последнем графике рис. 1. Мы можем выполнить аналогичное преобразование для лиц, сократив данные за счет использования лишь некоторых главных компонент и вернувшись затем в исходное пространство. Это возвращение в пространство исходных признаков можно выполнить с помощью метода **inverse_transform**.

Здесь мы визуализируем результаты реконструкции некоторых лиц, используя 10, 50, 100, 500 и 2000 компонент (рис. 9):

```
mglearn.plots.plot_pca_faces(X_train, X_test, image_shape)
plt.show()
```

Рис. 9 Реконструкция трех изображений лица с помощью постепенного увеличения числа главных компонент

Вы можете увидеть, что, когда мы используем лишь первые 10 главных компонент, фиксируется лишь общая суть картинки, например, ориентация лица и освещенность. По мере увеличения количества используемых компонент сохраняется все больше деталей изображения. Это соответствует включению большего числа слагаемых в сумму, показанную на рис. 8. Использование числа компонент, равного числу имеющихся

пикселей, означало бы, что мы, осуществив поворот, сохранили всю информацию и можем идеально реконструировать изображение.

Кроме того, мы можем применить **PCA** для визуализации всех лиц набора на диаграмме рассеяния, воспользовавшись первыми двумя главными компонентами (рис. 10). Для этого мы выделим классы, соответствующие лицам, с помощью определенного цвета и формы (аналогично тому, что делали для набора данных cancer):

```
mglearn.discrete_scatter(X_train_pca[:, 0], X_train_pca[:, 1], y_train)
plt.xlabel("Первая главная компонента")
plt.ylabel("Вторая главная компонента")
plt.show()
```

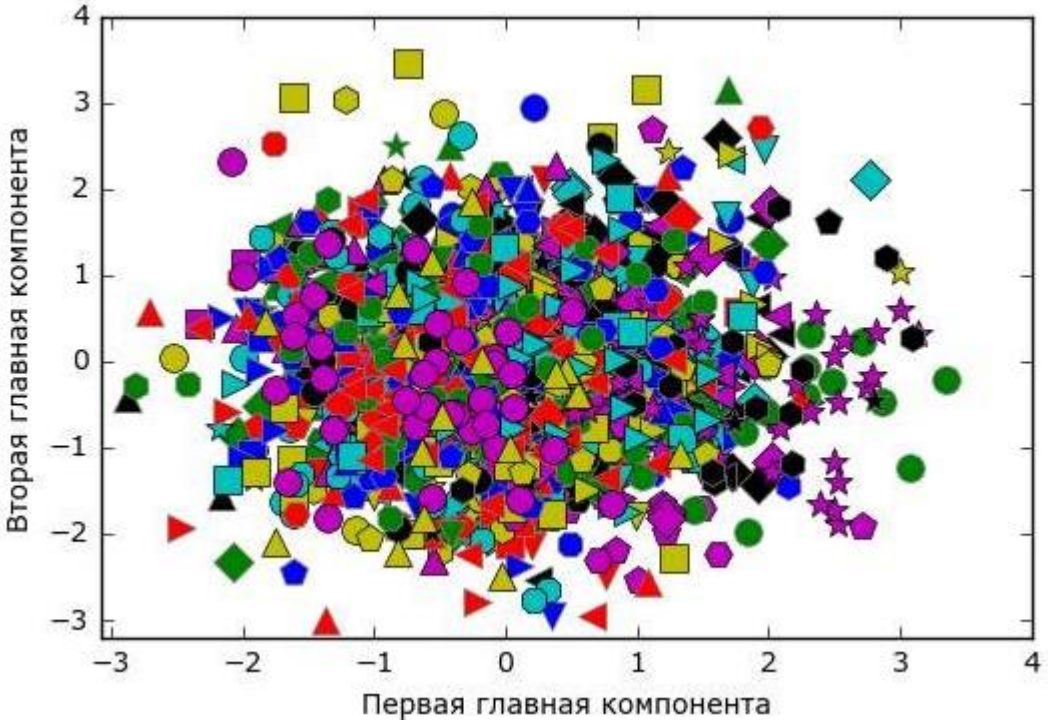

Рис. 10 Диаграмма рассеяния для набора лиц, использующая первые две главные компоненты (см. рис. 3 с соответствующим изображением для набора данных cancer)

Из рис. видно, когда мы используем лишь первые две главные компоненты, все данные представляют собой просто одно большое скопление данных без видимого разделения классов. Данный факт неудивителен, учитывая, что даже при использовании 10 компонент, как уже было показано ранее на рис. 9, **PCA** фиксирует самые общие характеристики лиц.

Код к лабораторной работе:

```
import mglearn
import sklearn
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

mglearn.plots.plot_pca_illustration()
plt.show()

from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

fig, axes = plt.subplots(15, 2, figsize=(10, 20))
malignant = cancer.data[cancer.target == 0]
benign = cancer.data[cancer.target == 1]
ax = axes.ravel()
for i in range(30):
    _, bins = np.histogram(cancer.data[:, i], bins=50)
    ax[i].hist(malignant[:, i], bins=bins, color=mglearn.cm3(0), alpha=.5)
    ax[i].hist(benign[:, i], bins=bins, color=mglearn.cm3(2), alpha=.5)
    ax[i].set_title(cancer.feature_names[i])
    ax[i].set_yticks(())
    ax[0].set_xlabel("Значение признака")
    ax[0].set_ylabel("Частота")
ax[0].legend(["доброкачественная", "злокачественная"], loc="best")
fig.tight_layout()
plt.show()


from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)

from sklearn.decomposition import PCA

# оставляем первые две главные компоненты
pca = PCA(n_components=2)
#  подгоняем  модель  PCA  на  наборе  данных  breast  cancer
pca.fit(X_scaled)
# преобразуем данные к первым двум главным компонентам
```

```
X_pca = pca.transform(X_scaled)
print("Форма исходного массива: {}".format(str(X_scaled.shape)))
print("Форма массива после сокращения размерности: {}".format(str(X_pca.shape)))



#  строим  график  первых  двух  главных  компонент,  классы  выделены  цветом
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], cancer.target)
plt.legend(cancer.target_names, loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("Первая главная компонента")
plt.ylabel("Вторая главная компонента")
plt.show()

print("форма главных компонент: {}".format(pca.components_.shape))

print("компоненты PCA:\n{}".format(pca.components_))

plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1], ["Первая компонента", "Вторая компонента"])
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)), cancer.feature_names, rotation=60, ha='left')
plt.xlabel("Характеристика")
plt.ylabel("Главные компоненты")
plt.show()

# conda install -c anaconda openssl
from sklearn.datasets import fetch_lfw_people

people = fetch_lfw_people(min_faces_per_person=20, resize=0.7)
image_shape = people.images[0].shape
fix, axes = plt.subplots(2, 5, figsize=(15, 8), subplot_kw={'xticks': (), 'yticks': ()})
for target, image, ax in zip(people.target, people.images, axes.ravel()):
    ax.imshow(image)
    ax.set_title(people.target_names[target])
plt.show()

print("форма массива изображений лиц: {}".format(people.images.shape))
print("количество классов: {}".format(len(people.target_names)))

# вычисляем частоту встречаемости каждого ответа
counts = np.bincount(people.target)
# печатаем частоты рядом с ответами
for i, (count, name) in enumerate(zip(counts, people.target_names)):
    print("{0:25} {1:3}".format(name, count), end=' ')
    if (i + 1) % 3 == 0:
        print()
```

```
mask = np.zeros(people.target.shape, dtype=np.bool)
for target in np.unique(people.target):
    mask[np.where(people.target == target)[0][:50]] = 1
X_people = people.data[mask]
y_people = people.target[mask]
#  для  получения  большей  стабильности  масштабируем  шкалу  оттенков  серого  так,  чтобы  значения
# были в диапазоне от 0 до 1 вместо использования шкалы значений от 0 до 255
X_people = X_people / 255.

from sklearn.neighbors import KNeighborsClassifier

# разбиваем данные на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(
    X_people, y_people, stratify=y_people, random_state=0)
#  строим  KNeighborsClassifier  с  одним  соседом
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("Правильность на тестовом наборе для 1-nn: {:.2f}".format(knn.score(X_test, y_test)))

mglearn.plots.plot_pca_whitening()
plt.show()

pca = PCA(n_components=100, whiten=True, random_state=0).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("обучающие данные после PCA: {}".format(X_train_pca.shape))

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
print("Правильность на тестовом наборе: {:.2f}".format(knn.score(X_test_pca, y_test)))

print("форма pca.components_: {}".format(pca.components_.shape))

# fix, axes = plt.subplots(3, 5, figsize=(15, 12),
#   subplot_kw={'xticks': (), 'yticks': ()})
# for i, (component, ax) in enumerate(zip(pca.components_, axes.ravel())):
#   ax.imshow(component.reshape(image_shape), cmap='viridis')
#   ax.set_title("{}. component".format((i + 1))
# plt.show()

mglearn.plots.plot_pca_faces(X_train, X_test, image_shape)
plt.show()

mglearn.discrete_scatter(X_train_pca[:, 0], X_train_pca[:, 1], y_train)
plt.xlabel("Первая главная компонента")
plt.ylabel("Вторая главная компонента")
plt.show()
```