Практическая работа 5. Задача понижения размерности и визуализации

1. Метод главных компонент PCA

Помимо кластеризации, к методам машинного обучения без учителя относятся методы визуализации, сжатия данных и поиск более информативного представления для дальнейшей обработки. Одним из самых простых и наиболее широко используемых алгоритмов для всех этих целей является метод главных компонент (principal component analysis).

На следующем рисунке показаны исходные точки данных, окрашенные в цвет, чтобы их можно было различать. Алгоритм PCA сначала находит направление максимальной дисперсии ("Component 1"). Это направление данных, которое содержит большую часть информации.

Затем алгоритм находит направление, которое содержит больше всего информации, но ортогонально первому направлению. В двух измерениях существует только одна возможная ориентация под прямым углом, но в пространствах более высоких измерений будет (бесконечно) много ортогональных направлений.

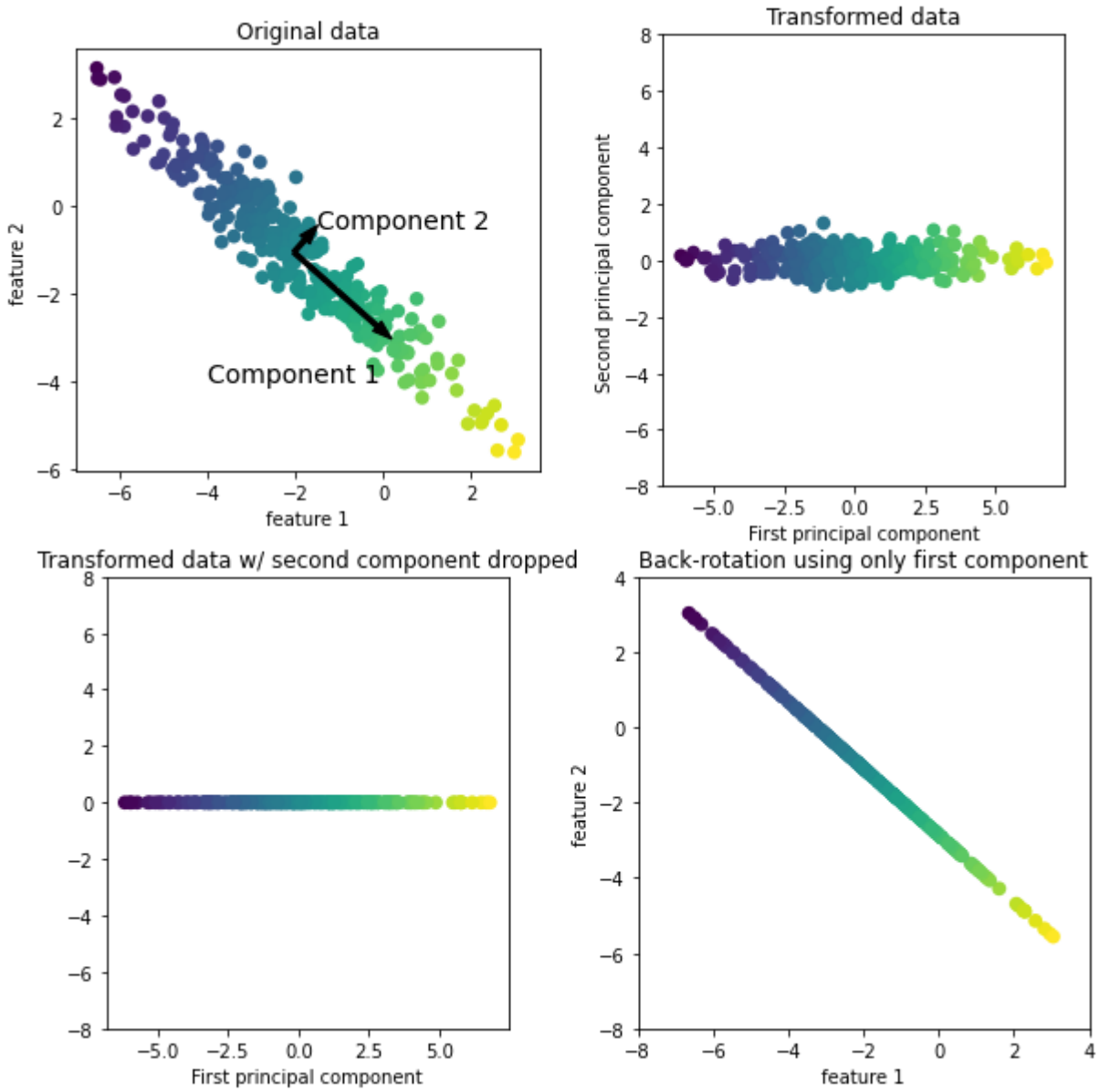
Найденные таким образом направления называются **главными компонентами**.

На втором графике показаны те же данные, но теперь они повернуты так, что первый главный компонент совпадает с осью x, а второй главный компонент совпадает с осью y. Среднее значение вычитается из каждого значения данных перед вращением, таким образом, преобразованные данные центрируются вокруг нуля.

Мы можем использовать PCA для уменьшения размера, сохранив только несколько основных компонентов. В этом примере мы можем оставить только первый главный компонент, как показано на третьем графике. Это уменьшит размерность данных: из двумерного массива данных мы получим одномерный массив данных. Однако следует отметить, что вместо того, чтобы оставлять только одну из начальных характеристик, мы находим наиболее интересное направление и оставляем его, то есть первую главную составляющую.

Наконец, мы можем отменить поворот и добавить среднее значение к значениям данных. В результате мы получаем данные, показанные на последнем графике. Эти точки расположены в пространстве начальных характеристик, но мы оставили только информацию, содержащуюся в первом основном компоненте. Это преобразование иногда используется, чтобы удалить эффект шума из данных или показать, какая часть информации сохраняется при использовании основных компонентов.

In [25]:
```
mglearn.plots.plot_pca_illustration()
```



Пример 1: PCA для набора данных Breast Cancer

В наборе данных Breast Cancer 30 признаков. Используя PCA, мы можем найти первые два основных компонента, и визуализировать данные в этом новом двумерном пространстве, чтобы "взглянуть" на них.

Прежде чем применять PCA, мы масштабируем наши данные так, чтобы каждая функция имела единичную дисперсию, используя **StandardScaler**:

In [26]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

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

Создадим экземпляр объекта PCA, найдем основные компоненты вызвав метод **fit**, а затем применим поворот и уменьшение размерности вызывая **transform**.

По умолчанию PCA только вращает данные, но сохраняет все основные компоненты. Чтобы уменьшить размерность данных, нам нужно указать сколько компонентов мы хотим сохранить при создании объекта PCA:

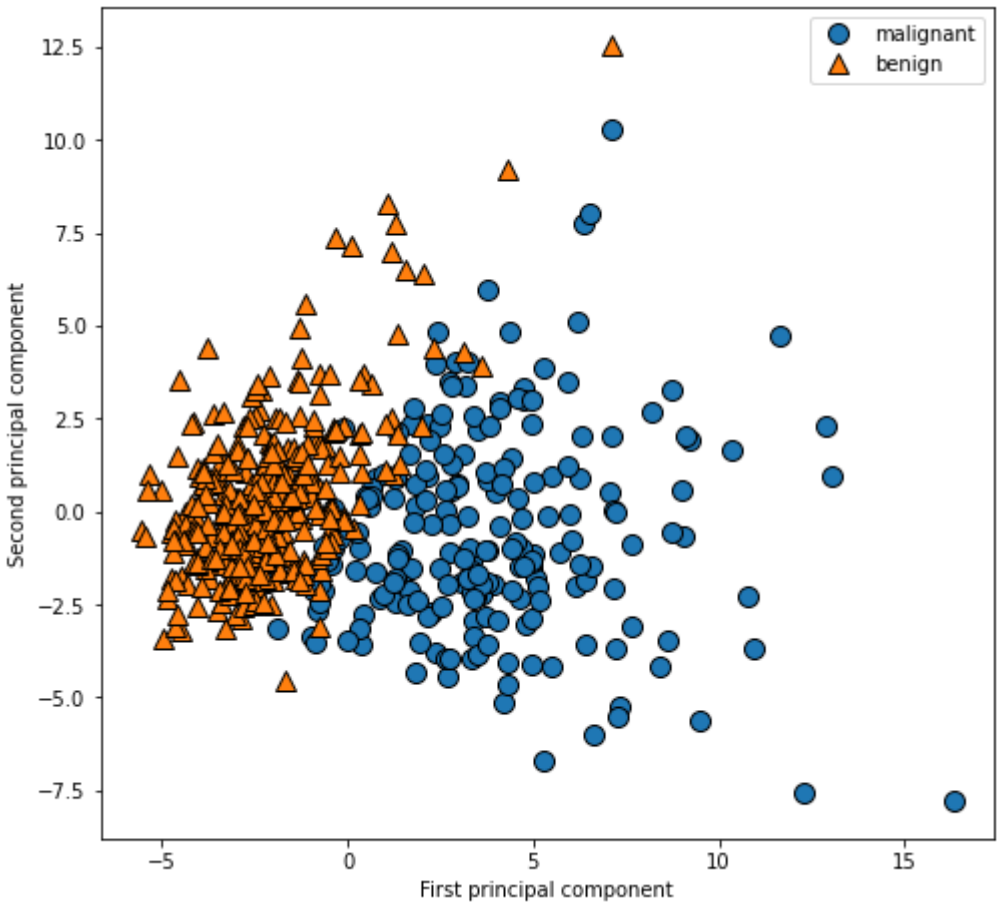
In [27]:
```
from sklearn.decomposition import PCA

# keep the first two principal components of the data
pca = PCA(n_components=2)

# fit PCA model to breast cancer data
pca.fit(X_scaled)

# transform data onto the first two principal components
X_pca = pca.transform(X_scaled)
print("Original shape: {}".format(str(X_scaled.shape)))
print("Reduced shape: {}".format(str(X_pca.shape)))
```

```
Original shape: (569, 30)
Reduced shape: (569, 2)
```

Построим теперь наши данные в плоскости их главных компонент:

In [28]:
```
# plot first vs. second principal component, colored by class
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("First principal component")
plt.ylabel("Second principal component")
```

Out[28]:
```
Text(0, 0.5, 'Second principal component')
```



Важно отметить, что PCA в процессе своей работы не использует информацию о классе, которую мы использовали здесь, чтобы раскрасить точки.

Можно видеть, что два класса довольно хорошо разделяются в этом двумерном пространстве. Мы также можем видеть, что злокачественные (синие) точки более распространены, чем доброкачественные (оранжевые) точки.

Недостатком PCA является то, что его главные компоненты часто не очень легко интерпретировать. Они представляют собой обычно очень сложные комбинации исходных признаков. Главные

компоненты сохраняются в атрибуте **components_** объекта PCA во время обучения:

In [29]:
```
print("PCA component shape: {}".format(pca.components_.shape))
```

```
PCA component shape: (2, 30)
```

Каждая строка в **components_** соответствует одному главному компоненту, и они отсортированы по важности (первый главный компонент идет первым и т. д.). Столбцы соответствуют исходному атрибуту функций PCA в этом примере: “mean radius,” “mean texture,” и так далее. Давайте посмотрим на содержимое **components_**:

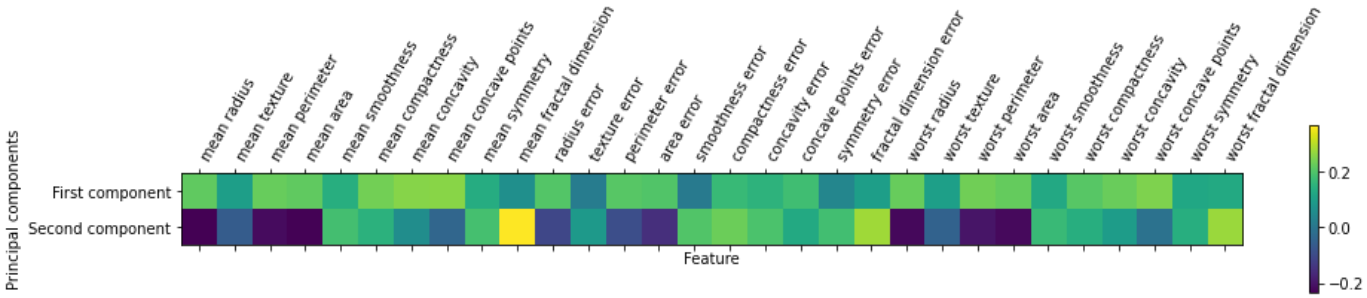
In [30]:
```
print("PCA components:\n{}".format(pca.components_))
```

```
PCA components:
[[ 0.21890244  0.10372458  0.22753729  0.22099499  0.14258969  0.23928535
   0.25840048  0.26085376  0.13816696  0.06436335  0.20597878  0.01742803
   0.21132592  0.20286964  0.01453145  0.17039345  0.15358979  0.1834174
   0.04249842  0.10256832  0.22799663  0.10446933  0.23663968  0.22487053
   0.12795256  0.21009588  0.22876753  0.25088597  0.12290456  0.13178394]
 [-0.23385713 -0.05970609 -0.21518136 -0.23107671  0.18611302  0.15189161
   0.06016536 -0.0347675   0.19034877  0.36657547 -0.10555215  0.08997968
  -0.08945723 -0.15229263  0.20443045  0.2327159   0.19720728  0.13032156
   0.183848    0.28009203 -0.21986638 -0.0454673  -0.19987843 -0.21935186
   0.17230435  0.14359317  0.09796411 -0.00825724  0.14188335  0.27533947]]
```

Можно визуализировать эти коэффициенты:

In [31]:
```
plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1], ["First component", "Second component"])
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)), cancer.feature_names, rotation=60, ha='left')
plt.xlabel("Feature")
plt.ylabel("Principal components")
```

Out[31]:
```
Text(0, 0.5, 'Principal components')
```



Пример 2: Применение PCA для уменьшения числа признаков в задаче классификации изображений Labeled Faces из набора Wild dataset

Загрузим несколько изображений из датасета, содержащего фото известных деятелей.

In [32]:
```
import matplotlib.pyplot as plt
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, cmap=plt.cm.gray)
    ax.set_title(people.target_names[target])
```

```
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976009
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976006
Downloading LFW data (~200MB): https://ndownloader.figshare.com/files/5976015
```

In [33]:
```
print("people.images.shape: {}".format(people.images.shape))
print("Number of classes: {}".format(len(people.target_names)))
```

```
people.images.shape: (3023, 87, 65)
Number of classes: 62
```

Как видим, у нас 62 класса (человека), изображений 3023, каждое размером 87x65. Посмотрим, сколько изображений есть для каждого деятеля. Есть некоторый перекос в сторону Дж. Буша.

In [34]:
```
import numpy as np

#count how often each target appears
counts = np.bincount(people.target)

# print counts next to target names
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()
```

```
Alejandro Toledo           39 Alvaro Uribe              35 Amelie Mauresmo           21
Andre Agassi               36 Angelina Jolie            20 Ariel Sharon              77
Arnold Schwarzenegger      42 Atal Bihari Vajpayee      24 Bill Clinton              29
Carlos Menem               21 Colin Powell             236 David Beckham             31
Donald Rumsfeld           121 George Robertson          22 George W Bush            530
Gerhard Schroeder         109 Gloria Macapagal Arroyo   44 Gray Davis                26
Guillermo Coria            30 Hamid Karzai              22 Hans Blix                 39
Hugo Chavez                71 Igor Ivanov               20 Jack Straw                28
Jacques Chirac             52 Jean Chretien             55 Jennifer Aniston          21
Jennifer Capriati          42 Jennifer Lopez            21 Jeremy Greenstock         24
Jiang Zemin                20 John Ashcroft             53 John Negroponte           31
Jose Maria Aznar           23 Juan Carlos Ferrero       28 Junichiro Koizumi         60
Kofi Annan                 32 Laura Bush                41 Lindsay Davenport         22
Lleyton Hewitt             41 Luiz Inacio Lula da Silva 48 Mahmoud Abbas             29
Megawati Sukarnoputri      33 Michael Bloomberg         20 Naomi Watts               22
Nestor Kirchner            37 Paul Bremer               20 Pete Sampras              22
Recep Tayyip Erdogan       30 Ricardo Lagos             27 Roh Moo-hyun              32
Rudolph Giuliani           26 Saddam Hussein            23 Serena Williams           52
Silvio Berlusconi          33 Tiger Woods               23 Tom Daschle               25
Tom Ridge                  33 Tony Blair               144 Vicente Fox               32
Vladimir Putin             49 Winona Ryder              24
```

Чтобы убрать такие перекосы, оставим только по (максимум) 50 изображений каждого класса.

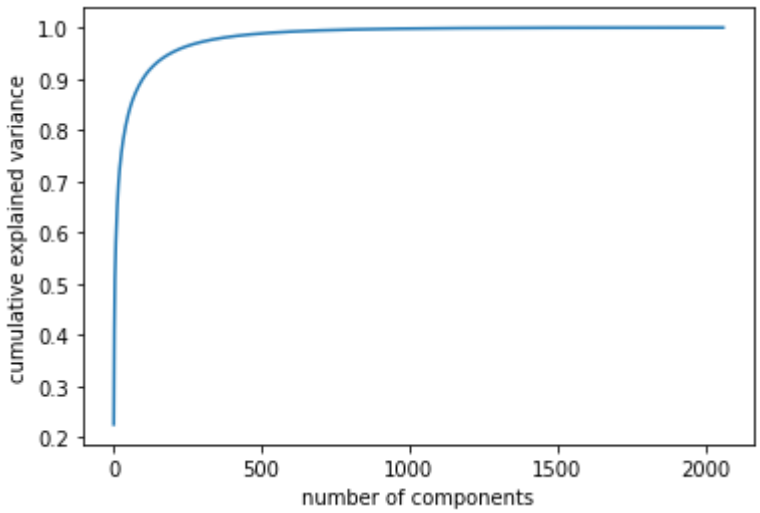
In [35]:
```
mask = np.zeros(people.target.shape, dtype=np.bool)
```

```
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]

# нормируем интенсивность [0,255] ->[0,1]
X_people = X_people / 255.
```

Выбор числа главных компонент в PCA преобразовании

Построим кривую, которая количественно показывает, какая часть общей 5655-мерной (87x65) дисперсии содержится в первых N компонентах. Глядя на этот график для многомерного набора данных, можно понять уровень избыточности, присутствующий в них.

In [36]:
```
from sklearn.decomposition import PCA

pca = PCA().fit(X_people)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



Мы видим, что нам потребуется около 100 компонентов, чтобы сохранить 90% дисперсии.

In [37]:
```
pca = PCA(0.90).fit(X_people)
pca.n_components_
```

Out[37]:
```
100
```

Разобъем данные на тестовую и обучающую выборку.

In [38]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_people, y_people, random_state=0)
```

In [39]:
```
X_train.shape
```

Out[39]:
```
(1547, 5655)
```

Обучим kNN классификатор на тренировочном наборе данных.

In [40]:
```
# build a KNeighborsClassifier using one neighbor
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("Test set score of 1-nn: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score of 1-nn: 0.28
```

Оставим с помощью PCA только 100 главных компонент.

In [41]:
```
from sklearn.decomposition import PCA

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("X_train_pca.shape: {}".format(X_train_pca.shape))
```

```
X_train_pca.shape: (1547, 100)
```

Вместо 5655 признаков теперь каждая точка (фото) характеризуется 100 признаками. Обучим вновь тот же классификатор на том же наборе тренировочных данных X_train_pca, прошедших PCA трансформацию. Классификатор обучается быстрее, т.к. признаков меньше. Что можно сказать о точности?

In [48]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
print("Test set accuracy: {:.2f}".format(knn.score(X_test_pca, y_test)))
```

```
Test set accuracy: 0.33
```

Как мы видим, точность улучшилась.

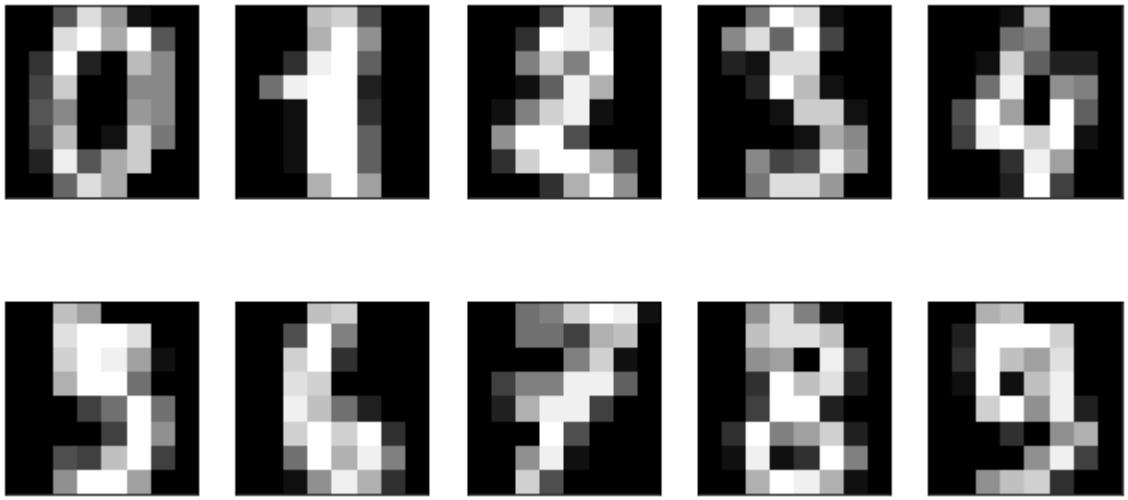
2. Метод t-SNE для визуализации данных

Хотя PCA может, как мы видели, эффективно понизить размерность данных, то, при понижении до 2-3, может использоваться и для визуализации данных. Однако существует класс алгоритмов визуализации, называемых алгоритмами множественного обучения, которые позволяют выполнять гораздо более сложные сопоставления между данными в исходном многомерном пространстве и пространстве малой размерности и часто обеспечивают лучшую визуализацию. Особенно полезен алгоритм t-SNE.

Идея t-SNE состоит в том, чтобы найти двумерное представление данных, которое максимально сохраняет расстояния между точками. t-SNE начинается со случайного двухмерного представления для каждой точки данных, а затем пытается сделать точки, которые находятся рядом в исходном пространстве признаков, ближе. Другими словами, он пытается сохранить информацию, указывающую, какие точки являются соседями друг друга.

Мы применим алгоритм обучения многообразия t-SNE к набору рукописных цифр, который включен в scikit-learn. Каждая точка данных в этом наборе данных представляет собой изображение в градациях серого 8 × 8 рукописной цифры от 0 до 1. На рисунке показан пример изображения для каждого класса:

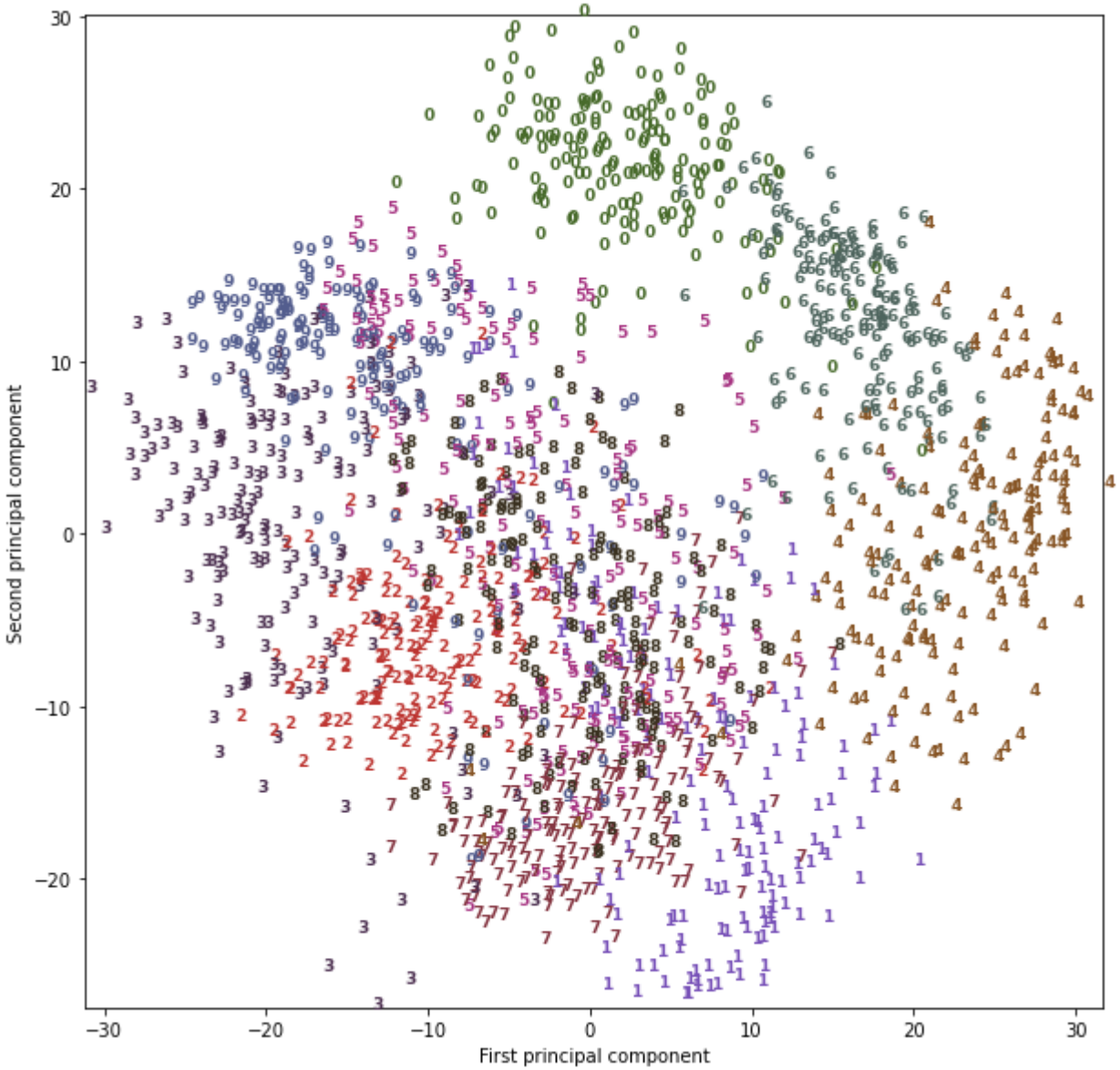
In [49]:
```
from sklearn.datasets import load_digits
digits = load_digits()

fig, axes = plt.subplots(2, 5, figsize=(10, 5),
                         subplot_kw={'xticks':(), 'yticks': ()})
for ax, img in zip(axes.ravel(), digits.images):
    ax.imshow(img, cmap=plt.cm.gray)
```



Давайте для начала воспользуемся методом PCA для визуализации данных, уменьшенных до двух измерений. Мы наносим на график первые два основных компонента и окрашиваем каждую точку в соответствии с ее классом:

In [50]:
```
#build a PCA model
pca = PCA(n_components=2)
pca.fit(digits.data)

# transform the digits data onto the first two principal components
digits_pca = pca.transform(digits.data)
colors = ["#476A2A", "#7851B8", "#BD3430", "#4A2D4E", "#875525",
          "#A83683", "#4E655E", "#853541", "#3A3120", "#535D8E"]

plt.figure(figsize=(10, 10))
plt.xlim(digits_pca[:, 0].min(), digits_pca[:, 0].max())
plt.ylim(digits_pca[:, 1].min(), digits_pca[:, 1].max())

for i in range(len(digits.data)):
    # actually plot the digits as text instead of using scatter
    plt.text(digits_pca[i, 0], digits_pca[i, 1], str(digits.target[i]),
             color = colors[digits.target[i]],
             fontdict={'weight': 'bold', 'size': 9})
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```

Out[50]:
```
Text(0, 0.5, 'Second principal component')
```

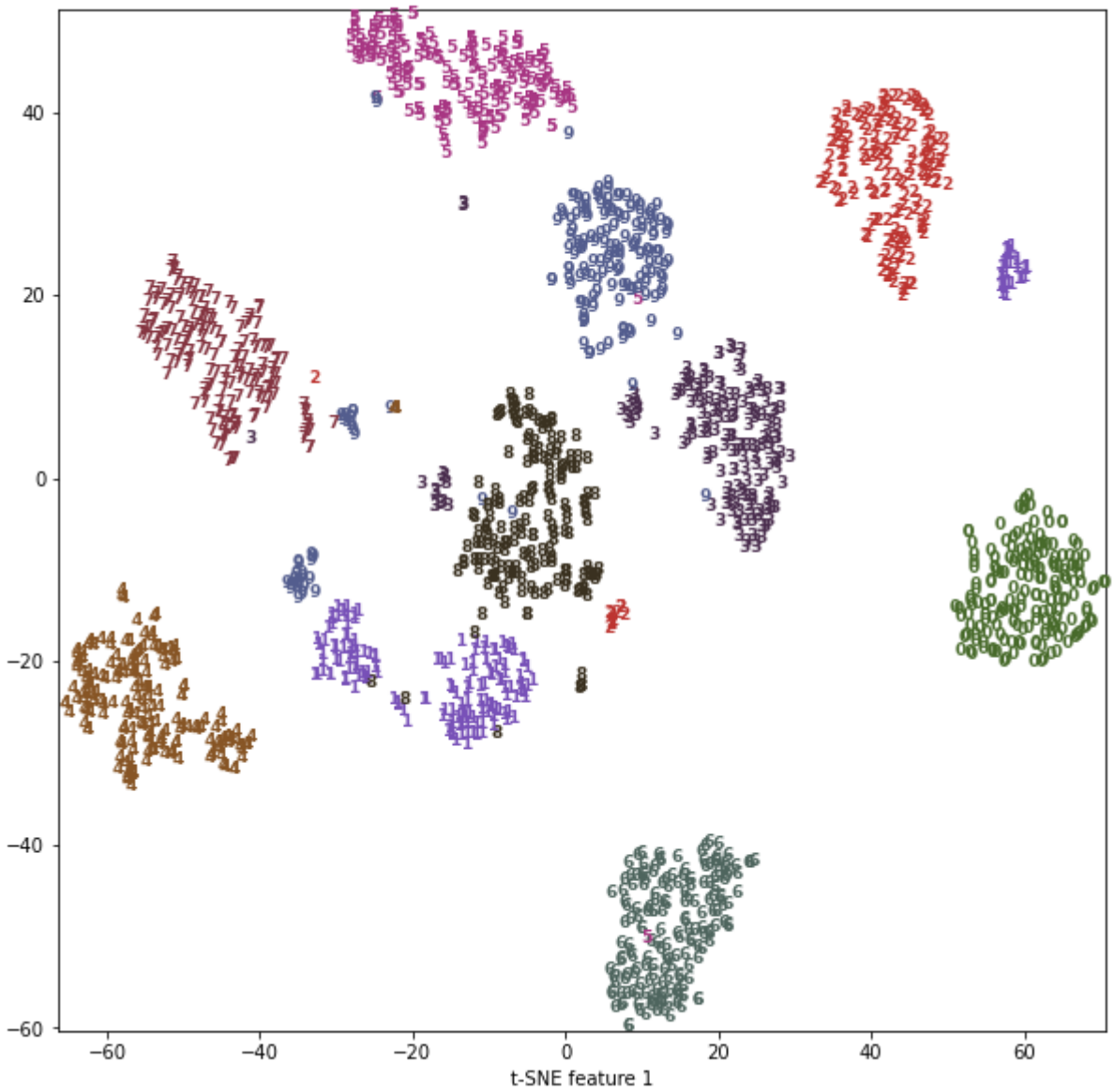


Здесь мы фактически использовали классы настоящих цифр только, чтобы показать, где находится соответсвующий класс. Цифры ноль, шесть и четыре относительно хорошо разделены с использованием первых двух основных компонентов, хотя они все еще перекрываются. Большинство других цифр значительно перекрываются.

Давайте применим теперь t-SNE к тому же набору данных и сравним результаты. Мы можем использовать метод fit_transform, который построит модель и сразу же вернет преобразованные данные:

In [51]:
```
from sklearn.manifold import TSNE
tsne = TSNE(random_state=42)

# use fit_transform instead of fit, as TSNE has no transform method
digits_tsne = tsne.fit_transform(digits.data)
plt.figure(figsize=(10, 10))
plt.xlim(digits_tsne[:, 0].min(), digits_tsne[:, 0].max() + 1)
plt.ylim(digits_tsne[:, 1].min(), digits_tsne[:, 1].max() + 1)

for i in range(len(digits.data)):
    # actually plot the digits as text instead of using scatter
    plt.text(digits_tsne[i, 0], digits_tsne[i, 1], str(digits.target[i]),
             color = colors[digits.target[i]],
             fontdict={'weight': 'bold', 'size': 9})
plt.xlabel("t-SNE feature 0")
plt.xlabel("t-SNE feature 1")
```

Out[51]:
```
Text(0.5, 0, 't-SNE feature 1')
```



Результат t-SNE очень впечатляет. Все классы довольно четко разделены. Почти все классы образуют единую плотную группу. Напомним, что метод t-SNE не знает меток классов: он полностью неконтролируемый. Тем не менее, он может найти представление данных в двух измерениях, которое четко разделяет классы, основываясь исключительно на том, насколько близки точки в исходном пространстве.