In [1]:
```
import numpy as np
import os

np.random.seed(42)


%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

Побудова датасету

Побудуємо "іграшковий" датасет у просторі розмірності 3:

In [2]:
```
np.random.seed(4)
m = 60
w1, w2 = 0.1, 0.3
noise = 0.1

angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
X = np.empty((m, 3))
X[:, 0] = np.cos(angles) + np.sin(angles)/2 + noise * np.random.r
andn(m) / 2
X[:, 1] = np.sin(angles) * 0.7 + noise * np.random.randn(m) / 2
X[:, 2] = X[:, 0] * w1 + X[:, 1] * w2 + noise * np.random.randn
(m)
```

PCA шляхом сингулярного розкладу (SVD decomposition)

In [3]:
```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]
```

In [4]:
```
m, n = X.shape

S = np.zeros(X_centered.shape)
S[:n, :n] = np.diag(s)
```

In [5]:
```
np.allclose(X_centered, U.dot(S).dot(Vt))
```

Out[5]:
```
True
```

In [6]:
```
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

In [7]:
```
X2D_using_svd = X2D
```

PCA використовуючи Scikit-Learn

In [8]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

In [9]:
```
X2D[:5]
```

Out[9]:
```
array([[ 1.26203346,  0.42067648],
       [-0.08001485, -0.35272239],
       [ 1.17545763,  0.36085729],
       [ 0.89305601, -0.30862856],
       [ 0.73016287, -0.25404049]])
```

In [10]:
```
X2D_using_svd[:5]
```

Out[10]:
```
array([[-1.26203346, -0.42067648],
       [ 0.08001485,  0.35272239],
       [-1.17545763, -0.36085729],
       [-0.89305601,  0.30862856],
       [-0.73016287,  0.25404049]])
```

Різниця отриманих проекцій лише у напрямку базисних векторів:

In [11]:
```
np.allclose(X2D, -X2D_using_svd)
```

Out[11]:
```
True
```

Відновимо елементи початкового датасету:

In [12]:
```
X3D_inv = pca.inverse_transform(X2D)
```

Звісно, з проекцією ми втратили частину інформації, отже таке "відновлення" не є точним, це просто відображення у початковий простір.

In [13]:
```
np.allclose(X3D_inv, X)
```

Out[13]:
```
False
```

Обрахуємо помилку відновлення:

In [14]:
```
np.mean(np.sum(np.square(X3D_inv - X), axis=1))
```

Out[14]:
```
0.010170337792848549
```

Відображення у початковий простір у термінах SVD виглядає як

In [15]:
```
X3D_inv_using_svd = X2D_using_svd.dot(Vt[:2, :])
```

Щоб отримати ідентичний результат, самостійно відцентруємо дані (sklearn робить це автоматично):

In [16]:
```
np.allclose(X3D_inv_using_svd, X3D_inv - pca.mean_)
```

Out[16]:
```
True
```

Головні компоненти:

In [17]:
```
pca.components_
```

Out[17]:
```
array([[-0.93636116, -0.29854881, -0.18465208],
       [ 0.34027485, -0.90119108, -0.2684542 ]])
```

Головні компоненти, отримані шляхом SVD:

In [18]:
```
Vt[:2]
```

Out[18]:
```
array([[ 0.93636116,  0.29854881,  0.18465208],
       [-0.34027485,  0.90119108,  0.2684542 ]])
```

Частка поясненої дисперсії:

In [19]:
```
pca.explained_variance_ratio_
```

Out[19]:
```
array([0.84248607, 0.14631839])
```

Втрата дисперсії:

In [20]:
```
1 - pca.explained_variance_ratio_.sum()
```

Out[20]:
```
0.011195535570688975
```

Пояснена дисперсія відповідає нормалізованим сингулярним числам у SVD:

In [21]:
```
np.square(s) / np.square(s).sum()
```

Out[21]:
```
array([0.84248607, 0.14631839, 0.01119554])
```

Візуалізуємо результат PCA:

In [22]:
```
from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d

class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwa
rgs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, rend
erer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)
```

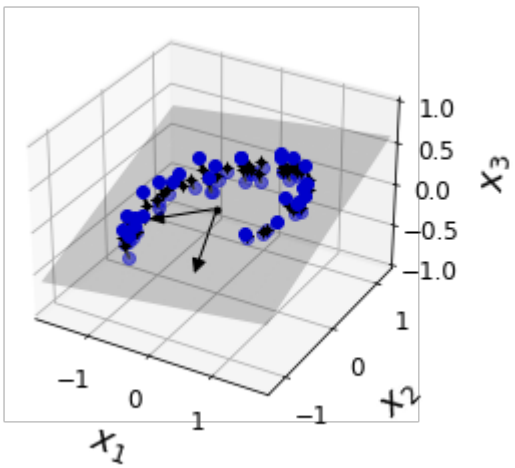
Задамо площину у термінах початкового базису:

In [23]:
```
axes = [-1.8, 1.8, -1.3, 1.3, -1.0, 1.0]

x1s = np.linspace(axes[0], axes[1], 10)
x2s = np.linspace(axes[2], axes[3], 10)
x1, x2 = np.meshgrid(x1s, x2s)

C = pca.components_
R = C.T.dot(C)
z = (R[0, 2] * x1 + R[1, 2] * x2) / (1 - R[2, 2])
```

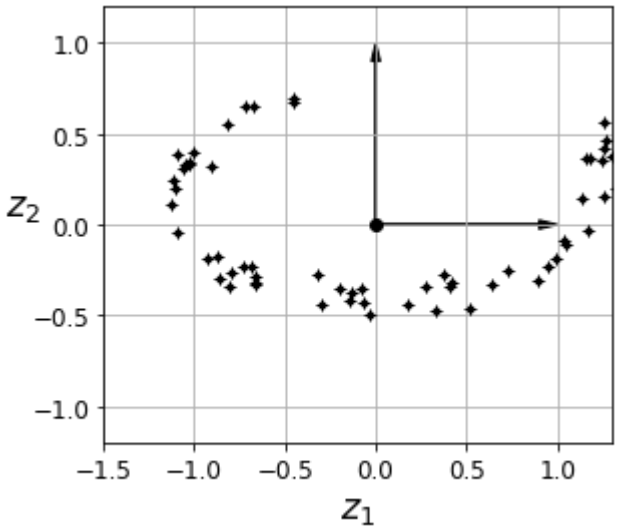
Зобразимо датасет, площину проекції та спроектовані точки

In [24]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(6, 3.8))
ax = fig.add_subplot(111, projection='3d')

X3D_above = X[X[:, 2] > X3D_inv[:, 2]]
X3D_below = X[X[:, 2] <= X3D_inv[:, 2]]

ax.plot(X3D_below[:, 0], X3D_below[:, 1], X3D_below[:, 2], "bo",
alpha=0.5)

ax.plot_surface(x1, x2, z, alpha=0.2, color="k")
np.linalg.norm(C, axis=0)
ax.add_artist(Arrow3D([0, C[0, 0]],[0, C[0, 1]],[0, C[0, 2]], mut
ation_scale=15, lw=1, arrowstyle="-|>", color="k"))
ax.add_artist(Arrow3D([0, C[1, 0]],[0, C[1, 1]],[0, C[1, 2]], mut
ation_scale=15, lw=1, arrowstyle="-|>", color="k"))
ax.plot([0], [0], [0], "k.")

for i in range(m):
    if X[i, 2] > X3D_inv[i, 2]:
        ax.plot([X[i][0], X3D_inv[i][0]], [X[i][1], X3D_inv
[i][1]], [X[i][2], X3D_inv[i][2]], "k-")
    else:
        ax.plot([X[i][0], X3D_inv[i][0]], [X[i][1], X3D_inv
[i][1]], [X[i][2], X3D_inv[i][2]], "k-", color="#505050")

ax.plot(X3D_inv[:, 0], X3D_inv[:, 1], X3D_inv[:, 2], "k+")
ax.plot(X3D_inv[:, 0], X3D_inv[:, 1], X3D_inv[:, 2], "k.")
ax.plot(X3D_above[:, 0], X3D_above[:, 1], X3D_above[:, 2], "bo")
ax.set_xlabel("$x_1$", fontsize=18, labelpad=10)
ax.set_ylabel("$x_2$", fontsize=18, labelpad=10)
ax.set_zlabel("$x_3$", fontsize=18, labelpad=10)
ax.set_xlim(axes[0:2])
ax.set_ylim(axes[2:4])
ax.set_zlim(axes[4:6])
```

Out[24]:
```
(-1.0, 1.0)
```



In [25]:
```
fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal')

ax.plot(X2D[:, 0], X2D[:, 1], "k+")
ax.plot(X2D[:, 0], X2D[:, 1], "k.")
ax.plot([0], [0], "ko")
ax.arrow(0, 0, 0, 1, head_width=0.05, length_includes_head=True,
head_length=0.1, fc='k', ec='k')
ax.arrow(0, 0, 1, 0, head_width=0.05, length_includes_head=True,
head_length=0.1, fc='k', ec='k')
ax.set_xlabel("$z_1$", fontsize=18)
ax.set_ylabel("$z_2$", fontsize=18, rotation=0)
ax.axis([-1.5, 1.3, -1.2, 1.2])
ax.grid(True)
```



PCA: ілюстрація

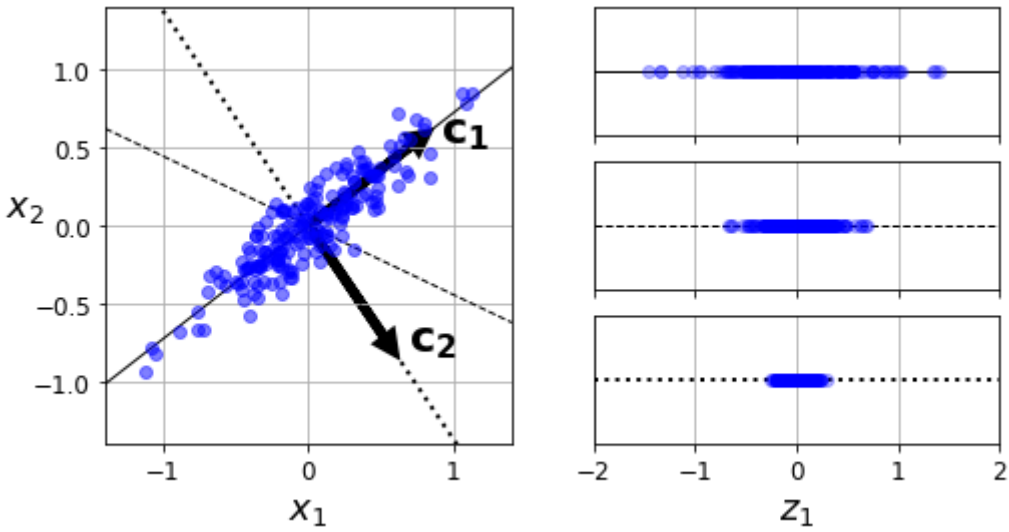
Проілюструємо розкид даних у різних проекціях

In [26]:
```
# створимо датасет
angle = np.pi / 5
stretch = 5
m = 200

np.random.seed(3)
X = np.random.randn(m, 2) / 10
X = X.dot(np.array([[stretch, 0],[0, 1]])) # розтяг
X = X.dot([[np.cos(angle), np.sin(angle)], [-np.sin(angle), np.co
s(angle)]]) # поворот
```

In [27]:
```
# напрямки

u1 = np.array([np.cos(angle), np.sin(angle)])
u2 = np.array([np.cos(angle - 2 * np.pi/6), np.sin(angle - 2 * n
p.pi/6)])
u3 = np.array([np.cos(angle - np.pi/2), np.sin(angle - np.pi/2)])
```

In [28]:
```
# проекції

X_proj1 = X.dot(u1.reshape(-1, 1))
X_proj2 = X.dot(u2.reshape(-1, 1))
X_proj3 = X.dot(u3.reshape(-1, 1))
```

In [29]:
```
plt.figure(figsize=(8,4))
plt.subplot2grid((3,2), (0, 0), rowspan=3)
plt.plot([-1.4, 1.4], [-1.4*u1[1]/u1[0], 1.4*u1[1]/u1[0]], "k-",
linewidth=1)
plt.plot([-1.4, 1.4], [-1.4*u2[1]/u2[0], 1.4*u2[1]/u2[0]], "k--",
linewidth=1)
plt.plot([-1.4, 1.4], [-1.4*u3[1]/u3[0], 1.4*u3[1]/u3[0]], "k:",
linewidth=2)
plt.plot(X[:, 0], X[:, 1], "bo", alpha=0.5)
plt.axis([-1.4, 1.4, -1.4, 1.4])
plt.arrow(0, 0, u1[0], u1[1], head_width=0.1, linewidth=5, length
_includes_head=True, head_length=0.1, fc='k', ec='k')
plt.arrow(0, 0, u3[0], u3[1], head_width=0.1, linewidth=5, length
_includes_head=True, head_length=0.1, fc='k', ec='k')
plt.text(u1[0] + 0.1, u1[1] - 0.05, r"$\mathbf{c_1}$", fontsize=2
2)
plt.text(u3[0] + 0.1, u3[1], r"$\mathbf{c_2}$", fontsize=22)
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$x_2$", fontsize=18, rotation=0)
plt.grid(True)

plt.subplot2grid((3,2), (0, 1))
plt.plot([-2, 2], [0, 0], "k-", linewidth=1)
plt.plot(X_proj1[:, 0], np.zeros(m), "bo", alpha=0.3)
plt.gca().get_yaxis().set_ticks([])
plt.gca().get_xaxis().set_ticklabels([])
plt.axis([-2, 2, -1, 1])
plt.grid(True)

plt.subplot2grid((3,2), (1, 1))
plt.plot([-2, 2], [0, 0], "k--", linewidth=1)
plt.plot(X_proj2[:, 0], np.zeros(m), "bo", alpha=0.3)
plt.gca().get_yaxis().set_ticks([])
plt.gca().get_xaxis().set_ticklabels([])
plt.axis([-2, 2, -1, 1])
plt.grid(True)

plt.subplot2grid((3,2), (2, 1))
plt.plot([-2, 2], [0, 0], "k:", linewidth=2)
plt.plot(X_proj3[:, 0], np.zeros(m), "bo", alpha=0.3)
plt.gca().get_yaxis().set_ticks([])
plt.axis([-2, 2, -1, 1])
plt.xlabel("$z_1$", fontsize=18)
plt.grid(True)
```



Стиснемо MNIST

In [30]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [31]:
```
from sklearn.decomposition import PCA
```

In [32]:
```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', version=1)
mnist.target = mnist.target.astype(np.uint8)
```

In [33]:
```
from sklearn.model_selection import train_test_split

X = mnist["data"]
y = mnist["target"]

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

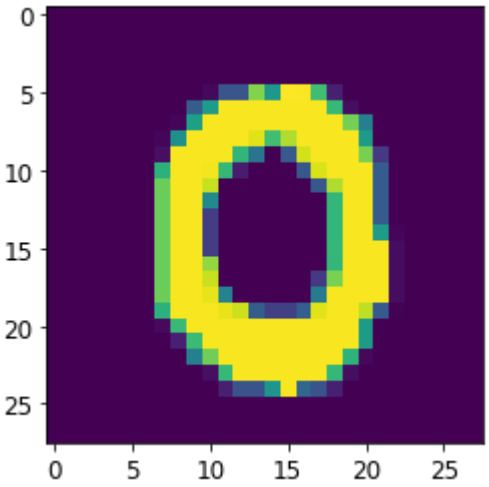
In [34]:
```
X_train.shape
```

Out[34]:
```
(52500, 784)
```

In [35]:
```
sample = X_train[0,:].reshape((28,28))
```

In [36]:
```
plt.imshow(sample)
```

Out[36]:
```
<matplotlib.image.AxesImage at 0x7fe715134a90>
```



In [37]:
```
y_train[0]
```

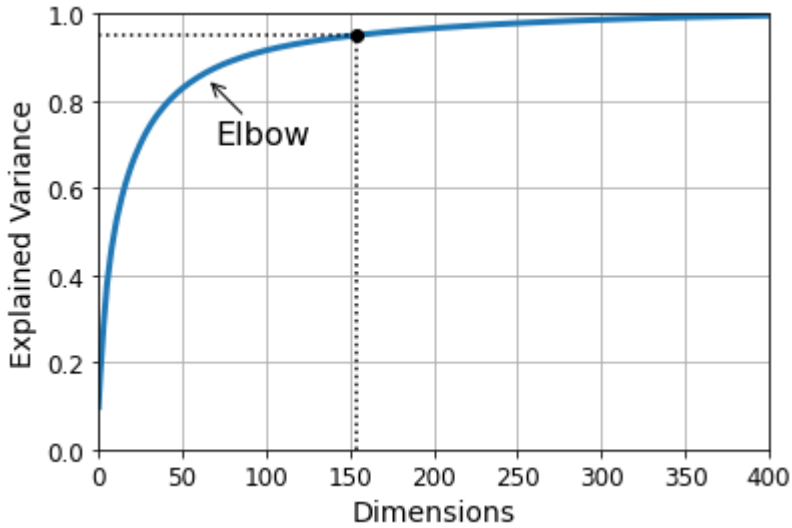
Out[37]:
```
0
```

In [38]:
```
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

In [39]:
```
d
```

Out[39]:
```
154
```

In [40]:
```
plt.figure(figsize=(6,4))
plt.plot(cumsum, linewidth=3)
plt.axis([0, 400, 0, 1])
plt.xlabel("Dimensions")
plt.ylabel("Explained Variance")
plt.plot([d, d], [0, 0.95], "k:")
plt.plot([0, d], [0.95, 0.95], "k:")
plt.plot(d, 0.95, "ko")
plt.annotate("Elbow", xy=(65, 0.85), xytext=(70, 0.7),
             arrowprops=dict(arrowstyle="->"), fontsize=16)
plt.grid(True)
plt.show()
```



In [41]:
```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

In [42]:
```
pca.n_components_
```

Out[42]:
```
154
```

In [43]:
```
np.sum(pca.explained_variance_ratio_)
```

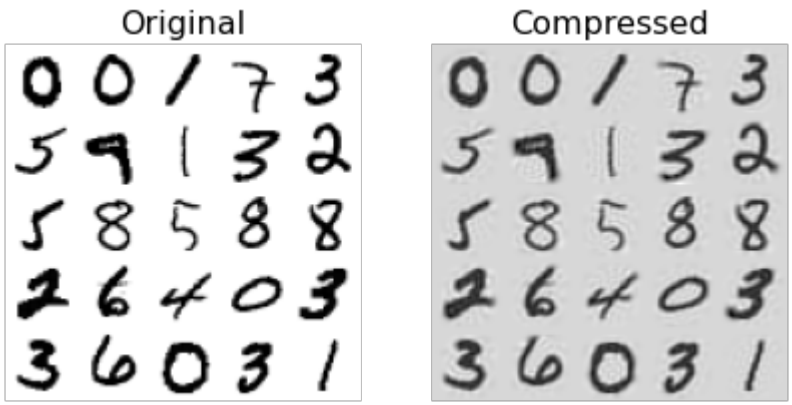
Out[43]:
```
0.9504334914295708
```

In [44]:
```
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

In [45]:
```
def plot_digits(instances, images_per_row=5, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instanc
es]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * image
s_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = mpl.cm.binary, **options)
    plt.axis("off")
```

In [46]:
```
plt.figure(figsize=(7, 4))
plt.subplot(121)
plot_digits(X_train[::2100])
plt.title("Original", fontsize=16)
plt.subplot(122)
plot_digits(X_recovered[::2100])
plt.title("Compressed", fontsize=16)

#save_fig("mnist_compression_plot")
```

Out[46]: Text(0.5, 1.0, 'Compressed')



In [47]:
```
X_reduced_pca = X_reduced
```

Incremental PCA

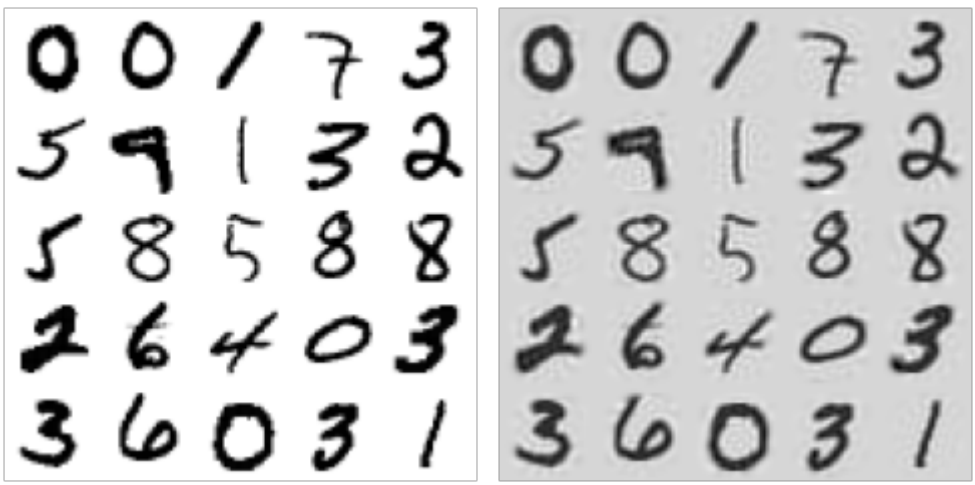
Incremental PCA - варіація PCA для датасетів великої розмірності.

In [48]:
```
from sklearn.decomposition import IncrementalPCA

n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
for X_batch in np.array_split(X_train, n_batches):
    print(".", end="")
    inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

```
....................................................................
................................
```

In [49]:
```
X_recovered_inc_pca = inc_pca.inverse_transform(X_reduced)
```

In [50]:
```
plt.figure(figsize=(7, 4))
plt.subplot(121)
plot_digits(X_train[::2100])
plt.subplot(122)
plot_digits(X_recovered_inc_pca[::2100])
plt.tight_layout()
```



In [51]:
```
X_reduced_inc_pca = X_reduced
```

Порівняємо PCA та incremental PCA. Середні значення однакові:

In [52]:
```
np.allclose(pca.mean_, inc_pca.mean_)
```

Out[52]:
```
True
```

Але результати стиснення - ні:

In [53]:
```
np.allclose(X_reduced_pca, X_reduced_inc_pca)
```

Out[53]:
```
False
```

Завдання

In [54]:
```
from sklearn.datasets import fetch_openml
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

In [55]:
```
np.random.seed(42)
```

In [56]:
```
X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
```

In [57]:
```
X.shape
```

Out[57]:
```
(70000, 784)
```

Завдання 1: Розбийте MNIST на тренувальну та навчальну вибірки (60k + 10k). Натренуйте Random Forest classifier ти виміряйте час тренування. Обчисліть точність на тестовій вибірці.

In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle
=False, train_size=60000)
```

In [59]:
```
rf = RandomForestClassifier(max_samples=0.8)
%time rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
CPU times: user 21.1 s, sys: 40.1 ms, total: 21.1 s
Wall time: 21.1 s
```

Out[59]:
```
0.9671
```

Завдання 2: Застосуйте PCA з поясненою дисперсією 95%. Повторіть тренування. Порівняйте час тренування та точніть з попередньою вправою.

In [60]:
```
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [61]:
```
X_train_pca.shape
```

Out[61]:
```
(60000, 154)
```

In [62]:
```
%time rf.fit(X_train_pca, y_train)

y_pred = rf.predict(X_test_pca)
accuracy_score(y_test, y_pred)
```

```
CPU times: user 57.6 s, sys: 233 ms, total: 57.8 s
Wall time: 57.6 s
```

Out[62]:
```
0.9483
```

More time, worse accuracy with PCA. It seems to me, that decision trees are trained faster with discrete data.

Завдання 3: Повторіть ці ж кроки з логістичною регресією. Зробіть висновки.

In [64]:
```
logreg = LogisticRegression(max_iter = 100)
%time logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
CPU times: user 1min 47s, sys: 34.8 s, total: 2min 22s
Wall time: 12.3 s

/home/daryna/anaconda3/envs/ml_ukma/lib/python3.7/site-packages/s
klearn/linear_model/_logistic.py:764: ConvergenceWarning: lbfgs f
ailed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opt
ions:
    https://scikit-learn.org/stable/modules/linear_model.html#log
istic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[64]:
```
0.9255
```

In [65]:
```
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [66]:
```
X_train_pca.shape
```

Out[66]:
```
(60000, 154)
```

In [67]:
```
%time logreg.fit(X_train_pca, y_train)

y_pred = logreg.predict(X_test_pca)
accuracy_score(y_test, y_pred)
```

```
CPU times: user 30.3 s, sys: 32.8 s, total: 1min 3s
Wall time: 5.54 s

/home/daryna/anaconda3/envs/ml_ukma/lib/python3.7/site-packages/s
klearn/linear_model/_logistic.py:764: ConvergenceWarning: lbfgs f
ailed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver opt
ions:
    https://scikit-learn.org/stable/modules/linear_model.html#log
istic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[67]:
```
0.9201
```

Less time, almost same accuracy with PCA, as it is supposed to be.

In [ ]:
```
```