

Отчёт по лабораторной работе №7 "Переобучение и регуляризация"

In [21]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from functions_from_sixth import compress

DATA_FILE_NAME_1 = 'Lab 7/ex7data1'
DATA_FILE_NAME_2 = 'Lab 7/ex7faces'
DATA_IMAGE_NAME = 'Lab 6/bird_small'
```

Imports from 'common' file:

In [2]:

```
import os
from scipy.io import loadmat

DATA_DIRECTORY = '../Data/'

def load_data_from_mat_file(filename, directory=DATA_DIRECTORY):
    filepath = directory + f'{filename}.mat'
    return loadmat(filepath)

def __convert_to_2d(X):
    try:
        X.shape[1]
        return X, False
    except IndexError:
        return np.array([X]).T, True

def normalize_features(X):
    delta = X.max(axis=0) - X.min(axis=0)
    average = X.sum(axis=0) / len(X)
    normalized = (X - average) / delta
    return normalized
```

1. Загрузите данные ex7data1.mat из файла.

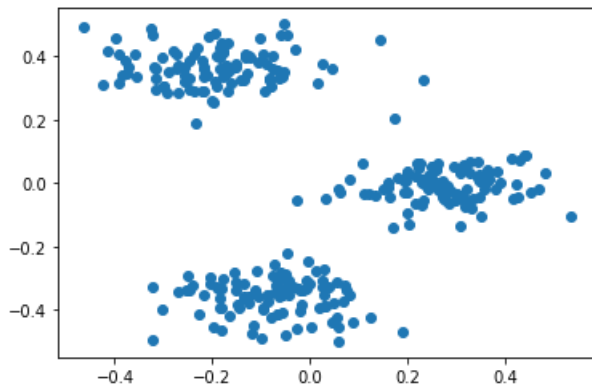
In [3]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_1)
X = pd.DataFrame({'x1': df_data['X'][:, 0], 'x2': df_data['X'][:, 1]})
X_norm = normalize_features(X.values)
```

2. Постройте график загруженного набора данных.

In [4]:

```
plt.plot(X_norm[:, 0], X_norm[:, 1], 'o')
plt.show()
```



3. Реализуйте функцию вычисления матрицы ковариации данных.

In [5]:

```
def get_covariance_matrix(X):
    return np.dot(X.T, X) / X.shape[0]
```

4. Вычислите координаты собственных векторов для набора данных с помощью сингулярного разложения матрицы ковариации (разрешается использовать библиотечные реализации матричных разложений).

In [6]:

```
def get_eigenvectors(X):
    Sigma = get_covariance_matrix(X)
    return np.linalg.svd(Sigma, full_matrices=False)
```

```
U, S, V = get_eigenvectors(X_norm)
print(U)
```

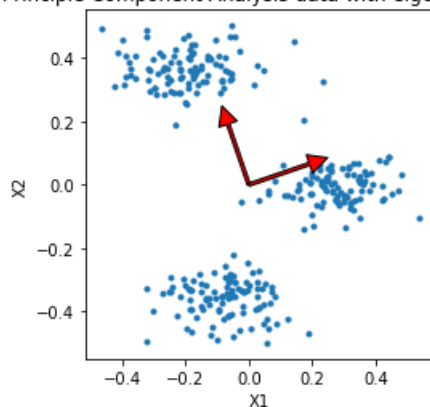
```
[[-0.32436342  0.94593254]
 [ 0.94593254  0.32436342]]
```

5. Постройте на графике из пункта 2 собственные векторы матрицы ковариации.

In [7]:

```
mu = X_norm.mean(axis=0)
projected_data = np.dot(X_norm, U)
variance = projected_data.std(axis=0).mean()
fig, ax = plt.subplots()
ax.plot(X_norm[:, 0], X_norm[:, 1], marker='o', linestyle="None", markersize=3)
for ind, axis in enumerate(U):
    start, end = mu, mu + variance * axis
    ax.annotate(
        '', xy=end, xycoords='data',
        xytext=start, textcoords='data',
        arrowprops=dict(facecolor='red', width=2.0))
ax.set_aspect('equal')
ax.set_title("Principle Component Analysis data with eigenvectors")
ax.set_xlabel('X1')
ax.set_ylabel('X2')
plt.show()
```

Principle Component Analysis data with eigenvectors



6. Реализуйте функцию проекции из пространства большей размерности в пространство меньшей размерности с помощью метода главных компонент.

In [9]:

```
def pca_transform(X, n_dimensions, U=None):
    U = get_eigenvectors(X)[0] if U is None else U
    return np.dot(X, U[:, :n_dimensions]), U
```

```
n_dimensions = 1
Z, U = pca_transform(X_norm, n_dimensions, U=U)
Z.shape, U.shape
```

Out[9]:

```
((300, 1), (2, 2))
```

7. Реализуйте функцию вычисления обратного преобразования.

In [10]:

```
def pca_reverse_transform(Z, U, n_dimensions):
    return np.dot(U[:, :n_dimensions], Z.T).T
```

```
X_approx = pca_reverse_transform(Z, U, n_dimensions)
```

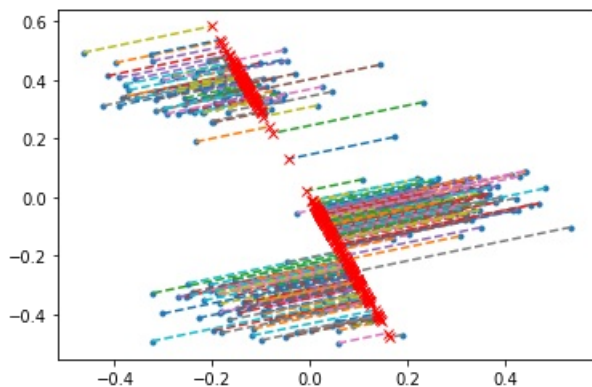
8. Постройте график исходных точек и их проекций на пространство меньшей размерности (с линиями проекций).

In [11]:

```
fig, ax = plt.subplots()
ax.plot(X_norm[:, 0], X_norm[:, 1], marker='o', linestyle="None", markersize=3)

for i, x in enumerate(X_norm):
    x_approx = X_approx[i]
    plt.plot([x[0], x_approx[0]], [x[1], x_approx[1]], '--')

plt.plot(X_approx[:, 0], X_approx[:, 1], 'rx')
plt.show()
```



9. Загрузите данные `ex7faces.mat` из файла.

In [12]:

```
df_data = load_data_from_mat_file(DATA_FILE_NAME_2)
X = df_data['X']
print('X shape:', X.shape)
```

X shape: (5000, 1024)

10. Визуализируйте 100 случайных изображений из набора данных.

In [13]:

```
X_rand = X[np.random.choice(X.shape[0], 100, replace=False), :]
fig, axs = plt.subplots(10, 10)
axs = axs.flatten()

for i, x in enumerate(X_rand):
    image = np.reshape(x, (32, 32), order="F")
    axs[i].imshow(image, cmap='gray')
    axs[i].xaxis.set_visible(False)
    axs[i].yaxis.set_visible(False)

plt.show()
```



11. С помощью метода главных компонент вычислите собственные векторы.

In [14]:

```
U, S, V = get_eigenvectors(X)
```

12. Визуализируйте 36 главных компонент с наибольшей дисперсией.

In [15]:

```
def plot_components(V, components_count):
    size = int(np.sqrt(components_count))
    fig, axs = plt.subplots(size, size, sharex=True, sharey=True, figsize=(10, 10))
    fig.suptitle(f"{components_count} PCA Eigenvectors of faces", fontsize=18)
    axs = axs.flatten()

    for i in range(components_count):
        image = np.reshape(V[i, :], (32, 32), order="F")
        axs[i].imshow(image, cmap='gray')
        axs[i].xaxis.set_visible(False)
        axs[i].yaxis.set_visible(False)

plot_components(V, 36)
plt.show()
```

36 PCA Eigenvectors of faces



13. Как изменилось качество выбранных изображений?

Качество изображений определенно стало хуже, однако всё ещё хорошо прослеживаются основные черты лиц. Это значит, что компоненты с наибольшей дисперсией отображают наиболее значимые признаки, например, такие как границы носа, глаз, рта, овала лица и т.д.

14. Визуализируйте 100 главных компонент с наибольшей дисперсией.

In [16]:

```
plot_components(V, 100)
plt.show()
```

100 PCA Eigenvectors of faces



15. Как изменилось качество выбранных изображений?

Исходя из результатов, можно заключить, что, чем меньше дисперсия компонента, тем хуже прослеживаются черты лица. Другими словами, чем меньше дисперсия компонента, тем менее важные признаки она отражает. Менее важные компоненты кодируют мелкие различия между лицами и шум.

16. Используйте изображение, сжатое в лабораторной работе №5.

In [22]:

```
df_data = load_data_from_mat_file(DATA_IMAGE_NAME)
A = df_data['A']
compressed_A = compress(A)
```

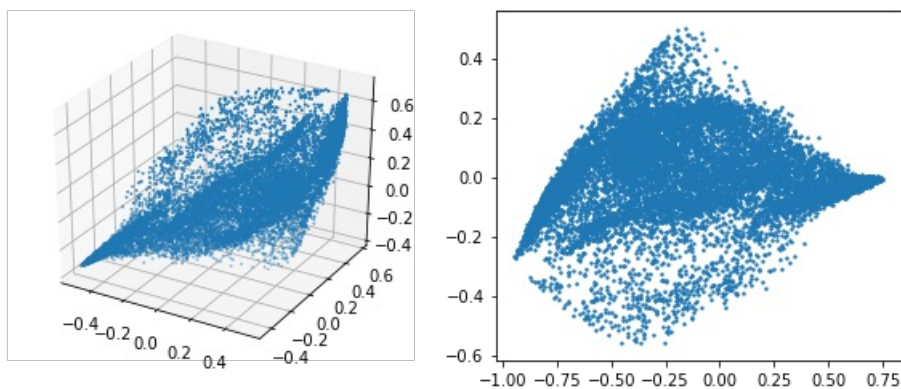
17. С помощью метода главных компонент визуализируйте данное изображение в 3D и 2D.

In [23]:

```
A_norm = normalize_features(A.reshape((A.shape[0] * A.shape[1], A.shape[2])))
Ax_reduced, U = pca_transform(A_norm, 2)
Ax_approx = pca_reverse_transform(Ax_reduced, U, 2)

fig = plt.figure(figsize=(10, 4))
ax = fig.add_subplot(1, 2, 1, projection='3d')
ax.scatter(xs=Ax_approx[:, 0], ys=Ax_approx[:, 1], zs=Ax_approx[:, 2], s=1)

ax2 = fig.add_subplot(1, 2, 2)
ax2.scatter(Ax_reduced[:, 0], Ax_reduced[:, 1], s=2)
plt.show()
```



18. Соответствует ли 2D изображение какой-либо из проекций в 3D?

После трансформации признаков изображения получилась плоскость. Она соответствует одной из проекций в 3D изображении. На рисунке слева 3D изображение повернуто таким образом, что можно убедиться в совпадении проекции в 3D и изображении в 2D.