```
import cv2
import os
import glob
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from numpy.linalg import inv
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, precision_score
```

```
2023-12-15 20:23:43.820417: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
2023-12-15 20:23:43.820674: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2023-12-15 20:23:43.821479: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2023-12-15 20:23:43.827576: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-12-15 20:25:44.821562: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
```

```
def build_GaborFilters():
    filters = []
    ksize = 28
    for theta in np.linspace(0, np.pi, 32):
        kern = cv2.getGaborKernel(
            (ksize, ksize), 4.0, theta, 10.0, 0.5, 0, ktype=cv2.CV_32F
        )
        kern /= 1.5 * kern.sum()
        filters.append(kern)
    return filters


def process(img, filters):
    accum = []
    for kern in filters:
        fimg = cv2.filter2D(img, cv2.CV_8UC3, kern)
        accum.append(fimg.reshape(-1))
    return np.vstack(accum).T
```

1

Consider a subset of the MNIST hand-written images which consists of the first 1000 images for digits 0 and 2. Each image is of size 28×28, and is transformed by Gabor wavelets to extract features at different orientations and scales. This results in an array of size $28 \times 28 \times 8 \times 4$ for each image, or equivalently, a matrix F of size 784×32. Each column of F corresponds to a different view of the image

```
(images, labels), (test_X, test_y) = mnist.load_data()
is_1_2 = (labels == 0) | (labels == 2)
labels = labels[is_1_2][:1000]
images = images[is_1_2][:1000]


f_list = []
for img in images:
    f = process(img, build_GaborFilters())
    f_list.append(f)
```

(2 points) Part 1:

Basic linear regression. Let x_k be the m-th view features of the k-th image, i.e., the m-th column of F_k. Let y_k be the label of the k-th image. Solve the following linear regression problem:

$$\min_w \sum_{k=1}^{K} (y_k - x_k^T w)^2$$

where K is the number of images in the subset. Report the classification accuracy for 10-fold cross-validation, where the subset is randomly divided into 10 equal parts, and each part is used as a test set once, while the remaining parts are used as a training set.

Let's denote $z_k = y_k - x_k^T w$ => $\min \sum_{k=1}^{K} (y_k - x_k^T w)^2 = \min z^T z$

$z = [y_1, .., y_k]^T - [x_1, ..., x_k]^T w = y - Xw$

$\min z^T z$ => $X^T(y - Xw) = 0$ => $w^* = (X^T X)^{-1} X^T y$

```
def get_w_star(X, y):
    return inv(X.T @ X) @ X.T @ y


kf = KFold(n_splits=10)
m = 10
for i, (train, test) in enumerate(kf.split(labels)):
    X_train = [f_list[img_indx][:, m] for img_indx in train]
    X_train = np.vstack(X_train).astype(float)

    X_test = [f_list[img_indx][:, m] for img_indx in test]
    X_test = np.vstack(X_test).astype(float)

    y_test = labels[test].astype(float)
    y_train = labels[train].astype(float)

    w_star = get_w_star(X_train, y_train)

    y_predict_test = np.round(X_test @ w_star)
    score_test = accuracy_score(y_test, y_predict_test)

    y_predict_train = np.round(X_train @ w_star)
    score_train = accuracy_score(y_train, y_predict_train)

    print(f"fold #{i} accuracy test {score_test} | accuracy train {score_train}")
```

```
fold #0 accuracy test 0.38 | accuracy train 1.0|
fold #1 accuracy test 0.39 | accuracy train 1.0|
fold #2 accuracy test 0.41 | accuracy train 1.0|
fold #3 accuracy test 0.43 | accuracy train 0.9988888888888889|
fold #4 accuracy test 0.45 | accuracy train 1.0|
fold #5 accuracy test 0.45 | accuracy train 1.0|
fold #6 accuracy test 0.42 | accuracy train 0.9988888888888889|
fold #7 accuracy test 0.3 | accuracy train 0.9988888888888889|
fold #8 accuracy test 0.51 | accuracy train 1.0|
fold #9 accuracy test 0.38 | accuracy train 1.0|
```

(6 points) Part 2: Robust linear regression.

Consider a variant of linear regression in which the input variables x_k are perturbed by some noise e_k with $||e_k||_2 \leq \delta$, where δ is a given parameter. The noise e_k represents the uncertainty or variability in the features extracted by the Gabor wavelets. The task is to seek a weight vector w that minimizes the worst-case error due to the noise e_k,i.e.,

$$\begin{aligned}
\min_w \max_{e_k} &\sum_{k=1}^{K} (y_k - (x_k + e_k)^T w)^2 \\
\text{s.t. } & ||e_k||^2_2 \leq \delta^2 \quad k = 1...K
\end{aligned}$$

Simplify the above problem to an unconstrained optimization problem. Solve the simplified problem and report the classification accuracy for 10-fold cross-validation.

$$\mathcal{L} = \sum_{k=1}^{K} (y_k - (x_k + e_k)^T w)^2 + \alpha(||e_k||_2^2 - \delta^2)$$

$$\frac{\partial \mathcal{L}}{\partial e_k} = 2w(y_k - w^T(x_k + e_k)) + 2\alpha e_k = 0$$

$$\Rightarrow w(scalar) + \alpha e_k = 0$$

$$\Rightarrow e_k \text{ is parallel to } w$$

$$\Rightarrow e_k = \lambda w$$

As far as a convex function reaches its maxima on the boundary (except constant functions) $||e_k||_2^2 = \delta^2$ => $e_k = \frac{\lambda}{||e||_2} \delta$ for $\lambda \in \{1, -1\}$

to maximize this function $(y_k - (x_k + e_k)^T w)^2 = (b - e_k^T w)^2 = (b - \lambda \delta ||w||_2)^2$ it is clear that the second term should have the sign as $(-b)$ => $\lambda = \frac{-b}{|b|}$

$$(b - \lambda \delta ||w||_2)^2 = (b + \frac{b}{|b|} \delta ||w||_2)^2 = b^2 + 2|b|\delta ||w||_2 + ||w||_2^2 =$$

$$= (|b| + \delta ||w||_2)^2 = (|y_k - x_k^T w| + \delta ||w||_2)^2$$

Back to the initial problem

$$\min_w \max_{e_k} \sum_{k=1}^{K} (y_k - (x_k + e_k)^T w) = \min_w \sum_{k=1}^{K} (|y_k - x_k^T w| + \delta ||w||_2)^2$$

rewrite to the simplified problem (because all terms are positive)

$$\min_w \sum_{k=1}^{K} (|y_k - x_k^T w| + \delta ||w||_2) = \min_w (||y - Xw||_1 + \delta K ||w||_2)$$

I tried to do something but it did not become better:

$$\frac{\partial f}{\partial w} = X^T sign(y - Xw') + \delta K \frac{w}{||w'||_2} = 0$$

$$K\delta \frac{w}{||w'||_2} = -X^T sign(y - Xw') = -X^T S$$

$$K\delta \frac{w}{||w'||_2} |_2 = K\delta => |X^T S|_2 = K\delta$$

So I came with the following:

$$K\delta \frac{w^*}{||w^*||_2} = -X^T sign(y - Xw^*)$$

$$|X^T sign(y - Xw^*)|_2 = \delta K$$

Did not come to closed-form solution

```
import cvxopy as cp
```

```
def get_w_star_cvx(X, y, delta):
    w = cp.Variable((784, 1))
    K = y.shape[0]
    cost = cp.norm(y - X @ w, 1) + delta * K * cp.norm(w, 2)
    prob = cp.Problem(cp.Minimize(cost))
    sol = prob.solve(solver=cp.ECOS)
    return w.value
```

```
kf = KFold(n_splits=10)
m = 10
for i, (train, test) in enumerate(kf.split(labels)):
    X_train = [f_list[img_indx][:, m] for img_indx in train]
    X_train = np.vstack(X_train).astype(float)

    X_test = [f_list[img_indx][:, m] for img_indx in test]
    X_test = np.vstack(X_test).astype(float)

    y_test = labels[test].astype(float)
    y_train = labels[train].astype(float)

    w_star = get_w_star_cvx(X_train, y_train, 0.4)

    y_predict_test = np.round(X_test @ w_star)
    score_test = accuracy_score(y_test, y_predict_test)

    y_predict_train = np.round(X_train @ w_star)
    score_train = accuracy_score(y_train, y_predict_train)

    print(f"fold #{i} accuracy test {score_test} | accuracy train {score_train}")
```

```
fold #0 accuracy test 0.81 | accuracy train 0.9511111111111111|
fold #1 accuracy test 0.74 | accuracy train 0.9533333333333334|
fold #2 accuracy test 0.63 | accuracy train 0.9488888888888889|
fold #3 accuracy test 0.77 | accuracy train 0.95|
fold #4 accuracy test 0.82 | accuracy train 0.9455555555555556|
fold #5 accuracy test 0.77 | accuracy train 0.9511111111111111|
fold #6 accuracy test 0.8 | accuracy train 0.9488888888888889|
fold #7 accuracy test 0.81 | accuracy train 0.95|
fold #8 accuracy test 0.81 | accuracy train 0.9422222222222222|
fold #9 accuracy test 0.75 | accuracy train 0.9522222222222222|
```

Add Gaussian noise with zero mean and standard deviation of 0.01 to x_k. Compare perform of the two linear regression methods for the noisy input data. x_k.

```
def generate_noise(X):
    return np.random.normal(loc=0, scale=0.01, size=(784, 32))
```

```
f_list_noisy = [f + generate_noise() for f in f_list]
```

```
print("w_star of initial problem")
kf = KFold(n_splits=10)
m = 10
for i, (train, test) in enumerate(kf.split(labels)):
    X_train = [f_list[img_indx][:, m] for img_indx in train]
    X_train = np.vstack(X_train).astype(float)

    X_test = [f_list_noisy[img_indx][:, m] for img_indx in test]
    X_test = np.vstack(X_test).astype(float)

    y_test = labels[test].astype(float)
    y_train = labels[train].astype(float)

    w_star = get_w_star(X_train, y_train)

    y_predict_test = np.round(X_test @ w_star)
    score_test = accuracy_score(y_test, y_predict_test)

    y_predict_train = np.round(X_train @ w_star)
    score_train = accuracy_score(y_train, y_predict_train)

    print(f"fold #{i} accuracy test {score_test} | accuracy train {score_train}")
```

```
w_star of initial problem
fold #0 accuracy test 0.37 | accuracy train 1.0|
fold #1 accuracy test 0.38 | accuracy train 1.0|
fold #2 accuracy test 0.41 | accuracy train 1.0|
fold #3 accuracy test 0.42 | accuracy train 0.9988888888888889|
fold #4 accuracy test 0.46 | accuracy train 1.0|
fold #5 accuracy test 0.44 | accuracy train 1.0|
fold #6 accuracy test 0.42 | accuracy train 0.9988888888888889|
fold #7 accuracy test 0.31 | accuracy train 0.9988888888888889|
fold #8 accuracy test 0.51 | accuracy train 1.0|
fold #9 accuracy test 0.39 | accuracy train 1.0|
```

```
print("w_star of second problem")
kf = KFold(n_splits=10)
m = 10
for i, (train, test) in enumerate(kf.split(labels)):
    X_train = [f_list_noisy[img_indx][:, m] for img_indx in train]
    X_train = np.vstack(X_train).astype(float)

    X_test = [f_list_noisy[img_indx][:, m] for img_indx in test]
    X_test = np.vstack(X_test).astype(float)

    y_test = labels[test].astype(float)
    y_train = labels[train].astype(float)

    w_star = get_w_star_cvx(X_train, y_train, delta=0.3)

    y_predict_test = np.round(X_test @ w_star)
    score_test = accuracy_score(y_test, y_predict_test)

    y_predict_train = np.round(X_train @ w_star)
    score_train = accuracy_score(y_train, y_predict_train)

    print(f"fold #{i} accuracy test {score_test} | accuracy train {score_train}")
```

```
w_star of second problem
fold #0 accuracy test 0.75 | accuracy train 0.9566666666666667|
fold #1 accuracy test 0.67 | accuracy train 0.9555555555555556|
fold #2 accuracy test 0.61 | accuracy train 0.9544444444444444|
fold #3 accuracy test 0.78 | accuracy train 0.9555555555555556|
fold #4 accuracy test 0.76 | accuracy train 0.9522222222222222|
fold #5 accuracy test 0.77 | accuracy train 0.9533333333333334|
fold #6 accuracy test 0.77 | accuracy train 0.9533333333333334|
fold #7 accuracy test 0.78 | accuracy train 0.9522222222222222|
fold #8 accuracy test 0.78 | accuracy train 0.9455555555555556|
fold #9 accuracy test 0.69 | accuracy train 0.9511111111111111|
```

The second method works tooo slow

Report classification accuracy for all views.

For first part solution:
fold #0 accuracy test 0.38 | accuracy train 1.0|
fold #1 accuracy test 0.39 | accuracy train 1.0|
fold #2 accuracy test 0.41 | accuracy train 1.0|
fold #3 accuracy test 0.43 | accuracy train 0.9988888888888889|
fold #4 accuracy test 0.45 | accuracy train 1.0|
fold #5 accuracy test 0.45 | accuracy train 1.0|
fold #6 accuracy test 0.42 | accuracy train 0.9988888888888889|
fold #7 accuracy test 0.3 | accuracy train 0.9988888888888889|
fold #8 accuracy test 0.51 | accuracy train 1.0|
fold #9 accuracy test 0.38 | accuracy train 1.0|

For second part solution: $\delta = 0.3$
fold #0 accuracy test 0.81 | accuracy train 0.9511111111111111|
fold #1 accuracy test 0.74 | accuracy train 0.9533333333333334|
fold #2 accuracy test 0.63 | accuracy train 0.9488888888888889|
fold #3 accuracy test 0.77 | accuracy train 0.95|
fold #4 accuracy test 0.82 | accuracy train 0.9455555555555556|
fold #5 accuracy test 0.77 | accuracy train 0.9511111111111111|
fold #6 accuracy test 0.8 | accuracy train 0.9488888888888889|
fold #7 accuracy test 0.81 | accuracy train 0.95|
fold #8 accuracy test 0.81 | accuracy train 0.9422222222222222|
fold #9 accuracy test 0.75 | accuracy train 0.9522222222222222|

With noize

first part solution: fold #0 accuracy test 0.37 | accuracy train 1.0|
fold #1 accuracy test 0.38 | accuracy train 1.0|
fold #2 accuracy test 0.41 | accuracy train 1.0|
fold #3 accuracy test 0.42 | accuracy train 0.9988888888888889|
fold #4 accuracy test 0.46 | accuracy train 1.0|
fold #5 accuracy test 0.44 | accuracy train 1.0|
fold #6 accuracy test 0.42 | accuracy train 0.9988888888888889|
fold #7 accuracy test 0.31 | accuracy train 0.9988888888888889|
fold #8 accuracy test 0.51 | accuracy train 1.0|
fold #9 accuracy test 0.39 | accuracy train 1.0|\

Second part solution: $\delta = 0.3$
fold #0 accuracy test 0.75 | accuracy train 0.9566666666666667|
fold #1 accuracy test 0.67 | accuracy train 0.9555555555555556|
fold #2 accuracy test 0.61 | accuracy train 0.9544444444444444|
fold #3 accuracy test 0.78 | accuracy train 0.9555555555555556|
fold #4 accuracy test 0.76 | accuracy train 0.9522222222222222|
fold #5 accuracy test 0.77 | accuracy train 0.9533333333333334|
fold #6 accuracy test 0.77 | accuracy train 0.9533333333333334|
fold #7 accuracy test 0.78 | accuracy train 0.9522222222222222|
fold #8 accuracy test 0.78 | accuracy train 0.9455555555555556|
fold #9 accuracy test 0.69 | accuracy train 0.9511111111111111|\