图像处理作业——图像分割

刘坤鑫*

2019 年 12 月 8 日

摘要

本文对高斯拉普拉斯 (LoG) 进行了数学推导；简要介绍了最小二乘法、多项式最小二乘法的原理，给出了一种基于矩阵运算的高效实现；剖析了 Python 第三方包 scikit-learn、scikit-image、scipy 源码，在此基础上实现了 RANSAC、霍夫变换算法；对于上述算法，以人工制造的数据和一张真实桌子图像 sobel 算子提取的边缘图像做对比分析。

1 LoG 的推导

二维高斯函数表达式如下：

$$G(x,y) = \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

拉普拉斯算子表达式如下：

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{2}$$

故高斯拉普拉斯 (LoG) 可推导如下：

$$\begin{aligned}
\nabla^2 G(x,y) &= \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2} \\
&= \frac{\partial}{\partial x}\left[\frac{-x}{\sigma^2}\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}\right] + \frac{\partial}{\partial y}\left[\frac{-y}{\sigma^2}\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}\right] \\
&= \left[\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right]\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} + \left[\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right]\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}} \\
&= \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}
\end{aligned} \tag{3}$$

*3017218061 软件工程一班

2 线性回归模型

2.1 最小二乘法

最小二乘法是回归分析中的一种标准方法，通过最小化每个方程的残差平方和来逼近超定方程组 (方程多于未知数的方程组) 的解。

考虑线性方程组：

$$\begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,m} \\ 1 & x_{2,1} & \dots & x_{2,m} \\ \dots & \dots & \dots & \dots \\ 1 & x_{n,1} & \dots & x_{n,m} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} \tag{4}$$

记为：

$$X \cdot w^T = y \tag{5}$$

其中 $x_{i,j}$ 表示第 i 个样本的第 j 元属性。最小二乘法试图确定最佳的 w 和 y，使得拟合出来的直线尽量符合样本数据。

考虑均方误差 (MSE) 作为性能的度量，我们试图优化：

$$\begin{aligned} \hat{w}^* &= \arg\min_{\hat{w}} (y - \hat{y})^2 \\ &= \arg\min_{\hat{w}} (y - X\hat{w})^2 \\ &= \arg\min_{\hat{w}} (y - X\hat{w})^{\mathrm{T}} (y - X\hat{w}) \end{aligned} \tag{6}$$

令 $E_{\hat{w}} = (y - \mathrm{X}\hat{w})^{\mathrm{T}}(y - \mathrm{X}\hat{w})$，对 $\hat{w}$ 求导得到：

$$\frac{\partial E_{\hat{w}}}{\partial \hat{w}} = 2X^T (X\hat{w} - y) \tag{7}$$

令上式为零可得 $\hat{w}$ 最优解的闭式解：

$$\hat{w}^* = \left(X^T X\right)^{-1} X^T y \tag{8}$$

这里为了简单起见，我们假定 $X^T X$ 为满秩矩阵 (full-rank matrix) 或正定矩阵 (positive definite matrix)，即 $(X^T X)^{-1}$ 有解。

求出 $\hat{w}$，y 也可以由式5得出。

Python 代码：

最小二乘法代码

```
class LinearLeastSquare(object):
    def fit(self, X, y):
        X = np.hstack((np.ones((X.shape[0], 1)), X))
```

```
        self.W = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
        return self

    def predict(self, X):
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        y = X.dot(self.W)
        return y

    def score(self, X, y):
        y_pred = self.predict(X)
        MSE = np.mean((y-y_pred)**2)
        return MSE
```

2.2 多项式最小二乘法

普通最小二乘法不能处理非线性的情况，但只要稍加改进，即可处理非线性的情况。
令

$$X^{(k)} = \begin{bmatrix} x_{1,1}^k & x_{1,2}^k & \dots & x_{1,m}^k \\ x_{2,1}^k & x_{2,2}^k & \dots & x_{2,m}^k \\ \dots & \dots & \dots & \dots \\ x_{n,1}^k & x_{n,2}^k & \dots & x_{n,m}^k \end{bmatrix} \tag{9}$$

$$w^{(k)} = \begin{bmatrix} w_{k*m+1} \\ w_{k*m+2} \\ \dots \\ w_{k*m+m} \end{bmatrix} \tag{10}$$

考虑方程组：

$$\sum_{k=0}^{D} X^{(k)} w^{(k)} = y \tag{11}$$

其中，D 为多项式最高次幂。
可见上式仍未线性方程组，可用普通最小二乘法或者任意一个线性回归模型解决。
Python 代码：

多项式最小二乘法代码

```
class PolynomialLeastSquares(object):
    def __init__(self, degree=3, base_estimator=LinearLeastSquare):
        self.degree = degree
        self.base_estimator = base_estimator()

    def fit(self, X, y):
        new_X = np.zeros(shape=(X.shape[0], 0))
        for i in range(self.degree):
            new_X = np.hstack((new_X, X**(i+1)))
        self.base_estimator.fit(new_X, y)
```

```
        self.W = self.base_estimator.W
        return self

    def predict(self, X):
        new_X = np.zeros(shape=(X.shape[0], 0))
        for i in range(self.degree):
            new_X = np.hstack((new_X, X**(i+1)))
        y = self.base_estimator.predict(new_X)
        return y

    def score(self, X, y):
        y_pred = self.predict(X)
        MSE = np.mean((y-y_pred)**2)
        return MSE
```

2.3 RANSAC

代码参考 sklearn 源码[1]。

RANSAC 代码

```
class RANSAC(object):
    def __init__(self,
                 base_estimator=LinearLeastSquare,
                 min_samples=None,
                 residual_threshold=None,
                 max_trials=100):
        self.base_estimator = base_estimator()
        self.min_samples = min_samples
        self.residual_threshold = residual_threshold
        self.max_trials = max_trials

    def fit(self, X, y):
        if self.min_samples is None:
            # assume linear model by default
            self.min_samples = X.shape[1] + 1

        if self.residual_threshold is None:
            # MAD (median absolute deviation)
            self.residual_threshold = np.median(np.abs(y - np.median(y)))

        n_inliers_best = 1
        score_best = np.inf
        inlier_mask_best = None
        X_inlier_best = None
        y_inlier_best = None

        sample_idxs = np.arange(X.shape[0])

        for i in range(self.max_trials):
            # choose random sample set
            all_idxs = np.arange(X.shape[0])
            np.random.shuffle(all_idxs)
            subset_idxs = all_idxs[:self.min_samples]
```

[1] https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/linear_model/_ransac.py

```
            # fit model for current random sample set
            self.base_estimator.fit(X[subset_idxs], y[subset_idxs])
            y_pred = self.base_estimator.predict(X)

            # residuals of all data for current random sample model
            residuals_subset = np.sum(np.abs(y-y_pred), axis=1)

            # classify data into inliers and outliers
            inlier_mask_subset = residuals_subset < self.residual_threshold
            n_inliers_subset = np.sum(inlier_mask_subset)

            # less inliers -> skip current random sample
            if n_inliers_subset < n_inliers_best:
                continue

            # extract inlier data set
            inlier_idxs_subset = sample_idxs[inlier_mask_subset]
            X_inlier_subset = X[inlier_idxs_subset]
            y_inlier_subset = y[inlier_idxs_subset]

            # score of inlier data set
            score_subset = self.base_estimator.score(
                X_inlier_subset, y_inlier_subset)

            # same number of inliers but worse score -> skip current random
            if (n_inliers_subset == n_inliers_best and score_subset > score_best):
                continue

            # save current random sample as best sample
            n_inliers_best = n_inliers_subset
            score_best = score_subset
            inlier_mask_best = inlier_mask_subset
            X_inlier_best = X_inlier_subset
            y_inlier_best = y_inlier_subset

        # estimate final model using all inliers
        self.base_estimator.fit(X_inlier_best, y_inlier_best)
        self.inlier_mask_ = inlier_mask_best
        return self

    def predict(self, X):
        return self.base_estimator.predict(X)

    def score(self, X, y):
        return self.base_estimator.score(X, y)
```

2.4 数据处理

按照题目要求，对于某条直线，y 轴数据服从高斯分布，即 x 轴也服从正态分布，然后人工添加离群点。代码如下：

数据处理

```
def make_data(n_samples=1000, n_inputs=1, n_outputs=1, noise=0.1, n_outliers=50):
    X = np.random.normal(size=(n_samples, n_inputs))
    W = np.ones(shape=(n_inputs, n_outputs))
```

```
    y = X.dot(W) + noise*np.random.normal(size=(n_samples, n_outputs))
    X[:n_outliers] = 3 + np.random.normal(size=(n_outliers, n_inputs))
    y[:n_outliers] = 0.5 + noise*np.random.normal(size=(n_outliers, n_outputs))
    return X, y
```

2.5 结果展示

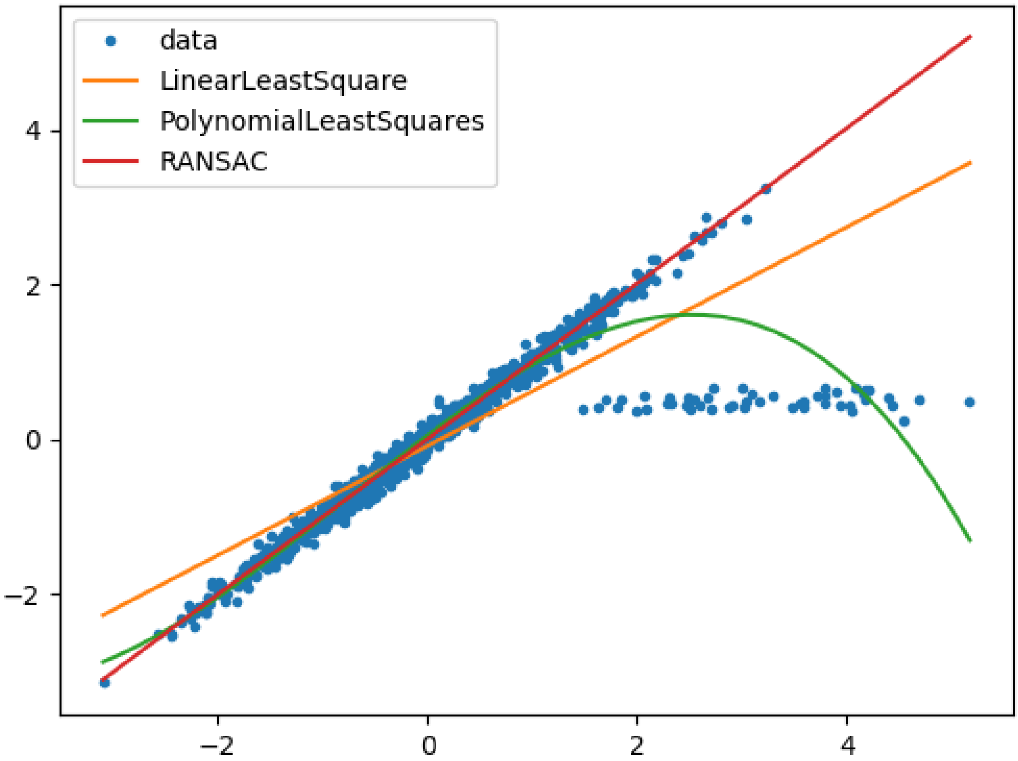


图 1: 三种线性回归模型拟合效果

如图1为三种线性回归模型拟合效果。其中，最小二乘法对于离群点敏感，多项式最小二乘法过拟合，RANSAC 有效解决了离群点的问题，增强了鲁棒性。

3 霍夫变换

3.1 原理

参考博文 “霍夫变换 - 疯狂奔跑 - 博客园”[2]。

[2] https://www.cnblogs.com/php-rearch/p/6760683.html

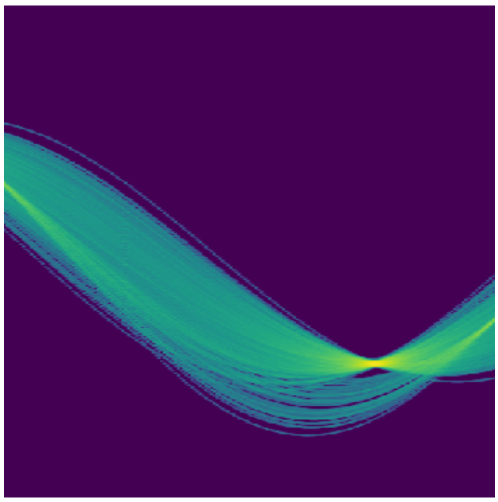
3.2 实现

参考 skimage 源码[3]

hough transform

```
def hough_line(img):
    # Rho and Theta ranges
    thetas = np.deg2rad(np.arange(-90.0, 90.0))
    width, height = img.shape
    diag_len = int(np.ceil(np.sqrt(width * width + height * height)))  # Dmax
    rhos = np.linspace(-diag_len, diag_len, diag_len * 2.0)
    # Cache some resuable values
    cos_t = np.cos(thetas)
    sin_t = np.sin(thetas)
    num_thetas = len(thetas)
    # Hough accumulator array of theta vs rho
    accumulator = np.zeros((2 * diag_len, num_thetas), dtype=np.uint64)
    y_idxs, x_idxs = np.nonzero(img) # (row, col) indexes to edges
    # Vote in the hough accumulator
    for i in range(len(x_idxs)):
        x = x_idxs[i]
        y = y_idxs[i]
        for t_idx in range(num_thetas):
            # Calculate rho. diag_len is added for a positive index
            rho = int(round(x * cos_t[t_idx] + y * sin_t[t_idx]) + diag_len)
            accumulator[rho, t_idx] += 1
    return accumulator, thetas, rhos
```

(a) 原始数据



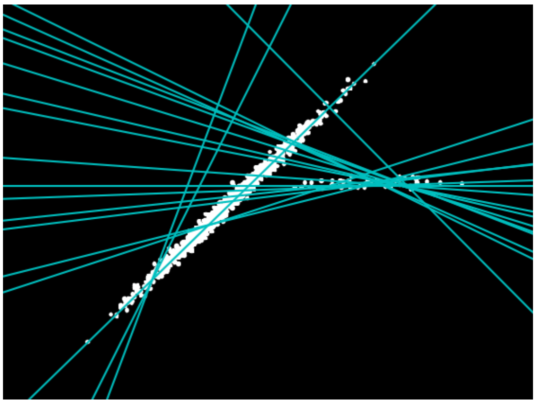
(b) 数据在霍夫空间

图 2: 霍夫变换

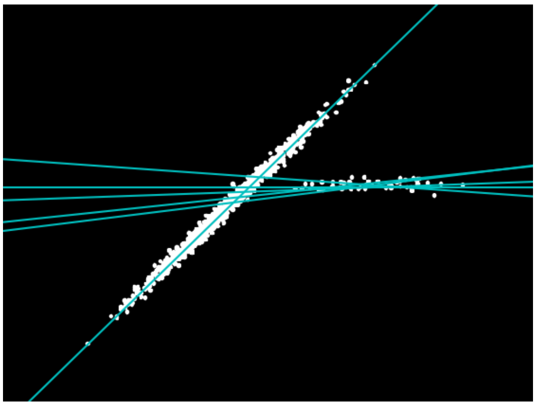
如图2为前文人工制造的数据。左图为原始数据，右图为数据在霍夫空间中的形态。因为霍夫变换适合在一张图像上进行操作，所以把题一的数据转换为图像格式。

由图可知，笛卡尔坐标中的一条直线代表霍夫空间中的一个点，霍夫空间中越亮的点，说明重叠次数越多，即在笛卡尔坐标中直线相交越多，这样的直线越有可能是要检测的边缘。

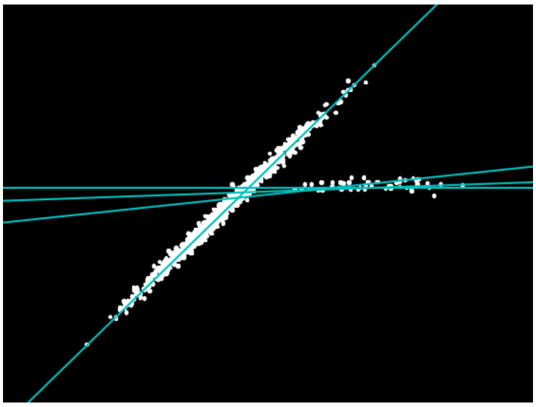
[3]https://github.com/scikit-image/scikit-image/blob/master/skimage/transform/_hough_transform.pyx



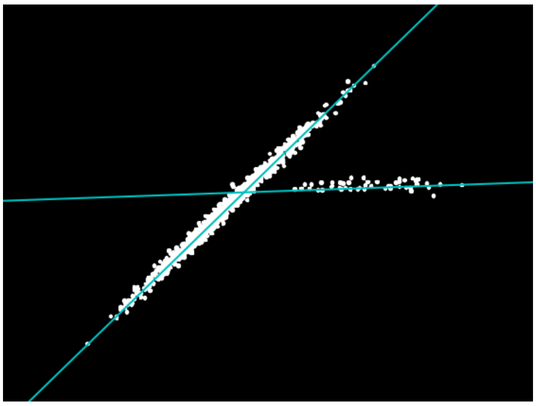
(a) threshold=50



(b) threshold=100



(c) threshold=130



(d) threshold=150

图 3: 霍夫直线不同的阈值的结果图

如图3为霍夫直线不同的阈值的结果图。在霍夫变换中，对于每个点的每个 θ 进行一次投票，最后可以得到计数的总和，计数越多，越有可能成为一条直线。阈值的作用在于筛选出计数大于阈值的直线。阈值越大，直线越少。

下面为求解霍夫变换空间和霍夫直线的代码：

hough space and hough line

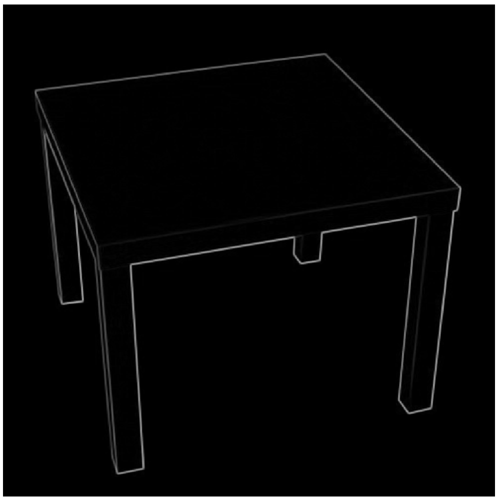
```
def show_transform(accumulator, path):
    image = np.log(1+accumulator)
    image = transform.resize(image, (512, 512))
    plt.imshow(image)
    plt.axis('off')
    plt.savefig(os.path.join('img', path))
    plt.close()


def show_line(image, accumulator, thetas, rhos, threshold, path):
    io.imshow(image)
    row, col = image.shape
    for _, angle, dist in zip(*transform.hough_line_peaks(accumulator, thetas, rhos, threshold=threshold)):
        y0 = (dist - 0 * np.cos(angle)) / np.sin(angle)
        y1 = (dist - col * np.cos(angle)) / np.sin(angle)
        plt.plot((0, col), (y0, y1), '-c')
    plt.axis((0, col, row, 0))
```

```
    path = os.path.join('img', path+str(threshold))
    plt.savefig(path)
    plt.close()
```

3.3 更多结果展示

(a) 桌子



(b) 桌子 sobel 算子边缘提取

图 4: 桌子

如图4为来源互联网的一张桌子图片。图右为经过 sobel 算子提取边缘后的图片。

sobel 算子如下：

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \tag{12}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{13}$$

Python 代码如下：

sobel 算子

```
edges = filters.sobel(image)
```

如图5为最小二乘法和 RANSAC 对桌子图片直线检测的结果。

先对图像进行二值化，然后抽离出数据点，再进行回归预测。然而，对于这种众多离群点的数据，回归算法往往不能取得好的效果。一个可行的解决方法是把图片分为若干个部分，分别进行回归预测，但由于复杂这里并未实现。

如图6为桌子图霍夫直线检测不同参数结果，效果良好。

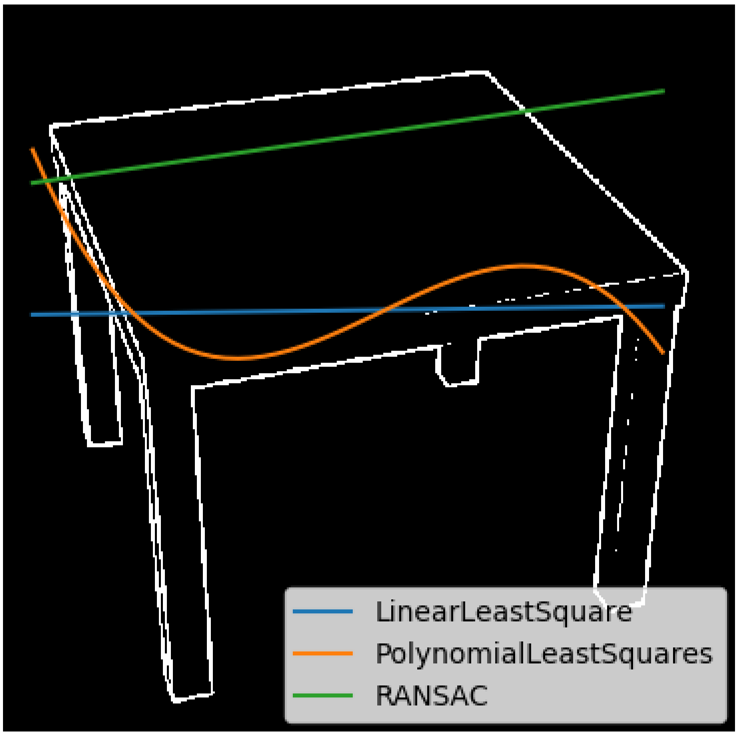
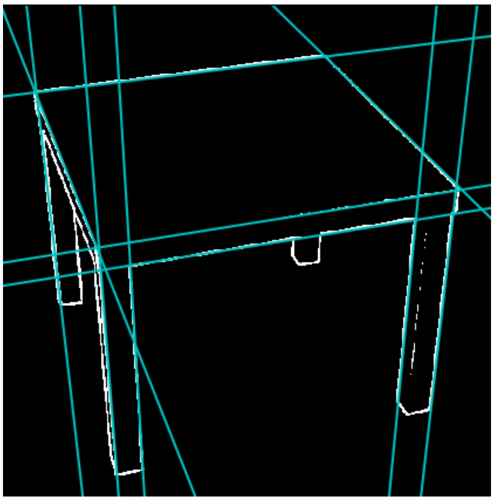
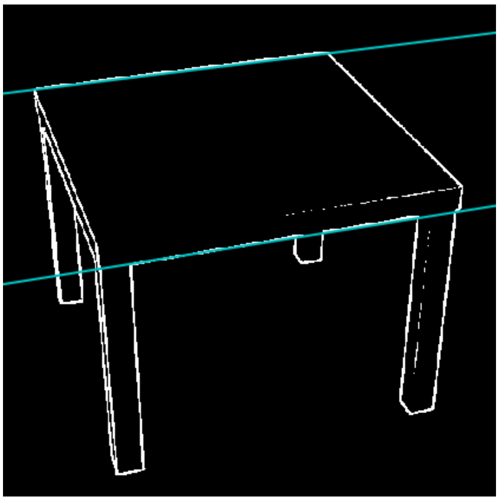


图 5: 桌子图线性回归模型直线检测



(a) threshold=100



(b) threshold=250

图 6: 桌子图霍夫直线检测

A 完整源码

LinearRegression.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(19260817)


def make_data(n_samples=1000, n_inputs=1, n_outputs=1, noise=0.1, n_outliers=50):
    X = np.random.normal(size=(n_samples, n_inputs))
    W = np.ones(shape=(n_inputs, n_outputs))
    y = X.dot(W) + noise*np.random.normal(size=(n_samples, n_outputs))
    X[:n_outliers] = 3 + np.random.normal(size=(n_outliers, n_inputs))
    y[:n_outliers] = 0.5 + noise*np.random.normal(size=(n_outliers, n_outputs))
    return X, y


class LinearLeastSquare(object):
    def fit(self, X, y):
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        self.W = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
        return self

    def predict(self, X):
        X = np.hstack((np.ones((X.shape[0], 1)), X))
        y = X.dot(self.W)
        return y

    def score(self, X, y):
        y_pred = self.predict(X)
        MSE = np.mean((y-y_pred)**2)
        return MSE


class PolynomialLeastSquares(object):
    def __init__(self, degree=3, base_estimator=LinearLeastSquare):
        self.degree = degree
        self.base_estimator = base_estimator()

    def fit(self, X, y):
        new_X = np.zeros(shape=(X.shape[0], 0))
        for i in range(self.degree):
            new_X = np.hstack((new_X, X**(i+1)))
        self.base_estimator.fit(new_X, y)
        self.W = self.base_estimator.W
        return self

    def predict(self, X):
        new_X = np.zeros(shape=(X.shape[0], 0))
        for i in range(self.degree):
            new_X = np.hstack((new_X, X**(i+1)))
        y = self.base_estimator.predict(new_X)
        return y

    def score(self, X, y):
        y_pred = self.predict(X)
        MSE = np.mean((y-y_pred)**2)
```

```
        return MSE


class RANSAC(object):
    def __init__(self,
                 base_estimator=LinearLeastSquare,
                 min_samples=None,
                 residual_threshold=None,
                 max_trials=100):
        self.base_estimator = base_estimator()
        self.min_samples = min_samples
        self.residual_threshold = residual_threshold
        self.max_trials = max_trials

    def fit(self, X, y):
        if self.min_samples is None:
            # assume linear model by default
            self.min_samples = X.shape[1] + 1

        if self.residual_threshold is None:
            # MAD (median absolute deviation)
            self.residual_threshold = np.median(np.abs(y - np.median(y)))

        n_inliers_best = 1
        score_best = np.inf
        inlier_mask_best = None
        X_inlier_best = None
        y_inlier_best = None

        sample_idxs = np.arange(X.shape[0])

        for i in range(self.max_trials):
            # choose random sample set
            all_idxs = np.arange(X.shape[0])
            np.random.shuffle(all_idxs)
            subset_idxs = all_idxs[:self.min_samples]

            # fit model for current random sample set
            self.base_estimator.fit(X[subset_idxs], y[subset_idxs])
            y_pred = self.base_estimator.predict(X)

            # residuals of all data for current random sample model
            residuals_subset = np.sum(np.abs(y-y_pred), axis=1)

            # classify data into inliers and outliers
            inlier_mask_subset = residuals_subset < self.residual_threshold
            n_inliers_subset = np.sum(inlier_mask_subset)

            # less inliers -> skip current random sample
            if n_inliers_subset < n_inliers_best:
                continue

            # extract inlier data set
            inlier_idxs_subset = sample_idxs[inlier_mask_subset]
            X_inlier_subset = X[inlier_idxs_subset]
            y_inlier_subset = y[inlier_idxs_subset]

            # score of inlier data set
            score_subset = self.base_estimator.score(
                X_inlier_subset, y_inlier_subset)
```

```
            # same number of inliers but worse score -> skip current random
            if (n_inliers_subset == n_inliers_best and score_subset > score_best):
                continue

            # save current random sample as best sample
            n_inliers_best = n_inliers_subset
            score_best = score_subset
            inlier_mask_best = inlier_mask_subset
            X_inlier_best = X_inlier_subset
            y_inlier_best = y_inlier_subset

        # estimate final model using all inliers
        self.base_estimator.fit(X_inlier_best, y_inlier_best)
        self.inlier_mask_ = inlier_mask_best
        return self

    def predict(self, X):
        return self.base_estimator.predict(X)

    def score(self, X, y):
        return self.base_estimator.score(X, y)


def main():
    X, y = make_data()
    plt.plot(X, y, linestyle='', marker='.', label='data')
    models = [
        LinearLeastSquare,
        PolynomialLeastSquares,
        RANSAC,
    ]
    for m in models:
        model = m()
        model.fit(X, y)
        X_test = np.linspace(X.min(), X.max())[:, np.newaxis]
        y_pred = model.predict(X_test)
        print(m.__name__, 'MSE:', model.score(X, y))
        plt.plot(X_test, y_pred, label=m.__name__)

    plt.legend(loc='upper left')
    plt.savefig(os.path.join('img', 'LinearRegression.png'))
    plt.show()


if __name__ == '__main__':
    main()
```

LSD.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, transform, data, filters
import LinearRegression as LR
np.random.seed(19260817)


def scatter2image(X, y):
    plt.scatter(X, y, color='black', marker='.')
    plt.axis('off')
```

```
    path = os.path.join('img', 'data.png')
    plt.savefig(path)
    plt.close()
    image = io.imread(path, as_gray=True)
    image = 1 - image
    io.imsave(path, image)
    return image


# http://blog.itpub.net/31077337/viewspace-2213246/
def hough_line(img):
    # Rho and Theta ranges
    thetas = np.deg2rad(np.arange(-90.0, 90.0))
    width, height = img.shape
    diag_len = int(np.ceil(np.sqrt(width * width + height * height)))  # Dmax
    rhos = np.linspace(-diag_len, diag_len, diag_len * 2.0)
    # Cache some resuable values
    cos_t = np.cos(thetas)
    sin_t = np.sin(thetas)
    num_thetas = len(thetas)
    # Hough accumulator array of theta vs rho
    accumulator = np.zeros((2 * diag_len, num_thetas), dtype=np.uint64)
    y_idxs, x_idxs = np.nonzero(img) # (row, col) indexes to edges
    # Vote in the hough accumulator
    for i in range(len(x_idxs)):
        x = x_idxs[i]
        y = y_idxs[i]
        for t_idx in range(num_thetas):
            # Calculate rho. diag_len is added for a positive index
            rho = int(round(x * cos_t[t_idx] + y * sin_t[t_idx]) + diag_len)
            accumulator[rho, t_idx] += 1
    return accumulator, thetas, rhos


# https://www.cnblogs.com/denny402/p/5158707.html
def show_transform(accumulator, path):
    image = np.log(1+accumulator)
    image = transform.resize(image, (512, 512))
    plt.imshow(image)
    plt.axis('off')
    plt.savefig(os.path.join('img', path))
    plt.close()


def show_line(image, accumulator, thetas, rhos, threshold, path):
    io.imshow(image)
    row, col = image.shape
    for _, angle, dist in zip(*transform.hough_line_peaks(accumulator, thetas, rhos, threshold=threshold)):
        y0 = (dist - 0 * np.cos(angle)) / np.sin(angle)
        y1 = (dist - col * np.cos(angle)) / np.sin(angle)
        plt.plot((0, col), (y0, y1), '-c')
    plt.axis((0, col, row, 0))
    path = os.path.join('img', path+str(threshold))
    plt.savefig(path)
    plt.close()


def test1():
    X, y = LR.make_data()
    image = scatter2image(X, y)
    accumulator, thetas, rhos = transform.hough_line(
```

```
        image)  # hough_line(image)
    show_transform(accumulator, 'hough_transform')
    show_line(image, accumulator, thetas, rhos, 50, 'hough_line')


def get_image():
    path = os.path.join('img', 'desk.png')
    image = io.imread(path, as_gray=True)
    edges = filters.sobel(image)
    io.imshow(edges)
    plt.savefig(os.path.join('img', 'desk_sobel.png'), dpi=300)
    plt.close()
    return edges


def image2scatter(image, threshold=0.1):
    row, col = image.shape
    X, y = [], []
    for i in range(row):
        for j in range(col):
            if image[i][j] > threshold:
                X.append([row-1-i])
                y.append(j)
    image = (image > threshold) * 1.0
    plt.imshow(image, plt.cm.gray)
    X = np.asarray(X).reshape((len(X), 1))
    y = np.asarray(y).reshape((len(y), 1))
    return X, y


def test2_1(image):
    X, y = image2scatter(image)
    models = [
        LR.LinearLeastSquare,
        LR.PolynomialLeastSquares,
        LR.RANSAC,
    ]
    for m in models:
        model = m()
        model.fit(X, y)
        X_test = np.linspace(X.min(), X.max())[:, np.newaxis]
        y_pred = model.predict(X_test)
        print(m.__name__, 'MSE:', model.score(X, y))
        plt.plot(X_test, y_pred, label=m.__name__)
    plt.legend(loc='lower right')
    plt.savefig(os.path.join('img', 'desk_LinearRegression.png'))

def test2_2(image):
    image = (image > 0.1) * 1.0
    accumulator, thetas, rhos = transform.hough_line(image)
    show_transform(accumulator, 'desk_hough_transform')
    show_line(image, accumulator, thetas, rhos, 250, 'desk_hough_line')


def test2():
    image = get_image()
    # test2_1(image)
    test2_2(image)


def main():
```

```python
    # test1()
    test2()


if __name__ == '__main__':
    main()
```