# 1. 题目

房价预测

用$n$维线性函数$\theta x^{(i)}+b$来回归房价$y^{(i)}$,其中$b$为常数,$\theta$为优化目标,$x^{(i)}$表示第$i$个样本,其中包含$n$个特征

$x^{(i)}$和$y^{(i)}$都经过归一化来统一量纲

$minL(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(\theta x^{(i)}+b-y^{(i)})$

# 2. 实现

本作业在jupyter notebook上完成,借用Kaggle平台,链接地址: https://www.kaggle.com/tianyilt/linear-regression-from-scratch-gradient-descent

jupyter notebook是一种python代码的集成开发环境,为数据科学的工作的所见即所得地创建分享提供支持,比如可以直接查看一个代码元胞的输出结果. 其中kaggle社区以jupyter为基础, 提供了在线jupyter环境, 免去了本地部署的问题, 任何人都可以直接上分享的网站上**鼠标点点点**就可以复现之前运行结果,极大地推动了社区的发展.

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files
in the input directory

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))

# Any results you write to the current directory are saved as output.

#Load the libraries and data...
import matplotlib.pyplot as plt
import matplotlib.animation as animation

data = pd.read_csv('../input/train.csv')


#Grab the relevant data, scale the predictor variable, and add a column of 1s for the
gradient descent...
```

```
x = data['GrLivArea']
y = data['SalePrice']

x = (x - x.mean()) / x.std()
x = np.c_[np.ones(x.shape[0]), x]

#GRADIENT DESCENT

alpha = 0.01 #Step size
iterations = 2000 #No. of iterations
m = y.size #No. of data points
np.random.seed(123) #Set the seed
theta = np.random.rand(2) #Pick some random values to start with


#GRADIENT DESCENT
def gradient_descent(x, y, theta, iterations, alpha):
    past_costs = []
    past_thetas = [theta]
    for i in range(iterations):
        prediction = np.dot(x, theta)
        error = prediction - y
        cost = 1/(2*m) * np.dot(error.T, error)
        past_costs.append(cost)
        theta = theta - (alpha * (1/m) * np.dot(x.T, error))
        past_thetas.append(theta)

    return past_thetas, past_costs

#Pass the relevant variables to the function and get the new values back...
past_thetas, past_costs = gradient_descent(x, y, theta, iterations, alpha)
theta = past_thetas[-1]

#Print the results...
print("Gradient Descent: {:.2f}, {:.2f}".format(theta[0], theta[1]))

#Plot the cost function...
plt.title('Cost Function J')
plt.xlabel('No. of iterations')
plt.ylabel('Cost')
plt.plot(past_costs)
plt.show()

#Animation

#Set the plot up,
fig = plt.figure()
ax = plt.axes()
plt.title('Sale Price vs Living Area')
plt.xlabel('Living Area in square feet (normalised)')
plt.ylabel('Sale Price ($)')
plt.scatter(x[:,1], y, color='red')
line, = ax.plot([], [], lw=2)
```

```
annotation = ax.text(-1, 700000, '')
annotation.set_animated(True)
plt.close()

#Generate the animation data,
def init():
    line.set_data([], [])
    annotation.set_text('')
    return line, annotation

# animation function.  This is called sequentially
def animate(i):
    x = np.linspace(-5, 20, 1000)
    y = past_thetas[i][1]*x + past_thetas[i][0]
    line.set_data(x, y)
    annotation.set_text('Cost = %.2f e10' % (past_costs[i]/10000000000))
    return line, annotation

anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=300, interval=0, blit=True)

anim.save('animation.gif', writer='imagemagick', fps = 30)


#Display the animation...
import io
import base64
from IPython.display import HTML

filename = 'animation.gif'

video = io.open(filename, 'r+b').read()
encoded = base64.b64encode(video)
HTML(data='''<img src="{0}" type="gif"
/>'''.format(encoded.decode('ascii')))
```

# 3. 参考资料


Linear Regression from scratch (Gradient Descent) | Kaggle

Andrew Ng's Coursera course