机器学习第二次实验 回归模型

- 姓名：管昀玫
- 学号：2013750
- 专业：计算机科学与技术

- 回归是监督学习的一个重要问题，回归用于预测输入变量和输出变量之间的关系，特别是当输入变量的值发生变化时，输出变量的值也随之发生变化。
- 回归模型是一种表示从输入变量到输出变量之间映射的函数
- 对连续值的预测
- 可以用合适的曲线揭示样本点随着自变量的变化关系

基本要求

将数据集winequality-white.csv按照4:1划分为训练集和测试集。

1. 构造线性回归模型，并采用批量梯度下降和随机梯度下降进行优化；输出训练集和测试集的均方误差（MSE），画出MSE收敛曲线。
2. 对于批量梯度下降和随机梯度下降，采用不同的学习率并进行MSE曲线展示，分析选择最佳的学习率。

特别需要注意：

- 划分数据集时尽可能保持数据分布的一致性，保持样本类别比例相似，可采用分层采样的方式。
- 需要对数据集进行一定的预处理

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedShuffleSplit
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv("winequality-white.csv")
df
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoho
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9
...	...	...	...	...	...	...	...	...	...	...	...
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoho
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8

4898 rows × 12 columns

In [3]:
```
# 中心化代码
def Normalization_fun(x):
    # 特征零均值
    x = (x - np.mean(x, 0)) / (np.max(x, 0) - np.min(x, 0))
    return x
```

In [4]:
```
def StratifiedSampling(df):
    df = df.sort_values(by="quality", ascending=True)

    # 将df根据quality进行分片
    df_layers = []
    for i in df["quality"].unique():
        df_layers.append(df[df["quality"] == i])
    train_index = []
    test_index = []

    # 按分片遍历
    for i in df_layers:
        test_use_list = []  # 新建一个list用于存储random结果
        while len(test_use_list) < int(len(i) / 5):  # 20%进测试集，80%进训练集
            test_use_list.append(np.random.randint(0, len(i)))  # 随机生成分片内的索引

        for ii in range(len(i)):  # 遍历分片内的索引
            if ii not in test_use_list:  # 如果索引不在测试集random索引中
                train_index.append(i.index[ii])  # 将索引加入训练集索引
            else:  # 否则加入测试集索引
                test_index.append(i.index[ii])
    # 将index索引转换为frame
    data = Normalization_fun(df.iloc[:, 0:-1])
    X_train = data.iloc[train_index, : ]
    Y_train = df.iloc[train_index, -1]
    X_test = data.iloc[test_index, : ]
    Y_test = df.iloc[test_index, -1]
    return X_train, Y_train, X_test, Y_test
```

In [5]:
```
def predict(alpha, beta, x):
    arr = alpha * x
    return np.sum(arr) + beta
```

In [6]:
```
def gradient_descent(x, y, alpha, beta, learn_rate):# 批量梯度下降
    # gradient_arr是整个 alpha 偏导数数组
    gradient_arr = np.zeros((1, x.shape[1]))
    gradient_beta = 0
    mean_s_err = 0
    for line in range(x.shape[0]):
```

```
        xline = x.iloc[line, :]
        yline = y.iloc[line]
        # err = y - (alpha X + beta)
        err = yline - predict(alpha, beta, xline)
        # print(gradient_arr,"\n", xline)
        gradient_arr += err * xline.values
        gradient_beta += err
        mean_s_err += err ** 2

    # arr 是 alpha vector的梯度vec,  alpha0 是 arr[0]
    gradient_arr = gradient_arr * 2 / x.shape[0]
    gradient_beta = gradient_beta * 2 / x.shape[0]
    mean_s_err /= x.shape[0]

    alpha += np.reshape(gradient_arr, alpha.shape) * learn_rate
    beta += gradient_beta * learn_rate
    return alpha, beta, mean_s_err
```

In [7]:

```
def gradient_descent_random(x, y, alpha, beta, learn_rate): # 随机梯度下降
    randomId = int(np.random.random_sample() * x.shape[0])
    x = x.iloc[randomId, :]
    y = y.iloc[randomId]
    gradient_arr = np.zeros(x.shape[0])
    gradient_beta = 0
    err = y - predict(alpha, beta, x)
    gradient_arr += err * x
    gradient_beta += err

    # arr 是 alpha vector的梯度vec,  alpha0 是 arr[0]
    gradient_arr = gradient_arr * 2
    gradient_beta = gradient_beta * 2

    alpha += np.reshape(gradient_arr, alpha.shape) * learn_rate
    beta += gradient_beta * learn_rate

    return alpha, beta
```

In [8]:

```
# 岭回归
def gradient_descent_random_L2(x, y, alpha, beta, learn_rate, L2_lambda):
    randomId = int(np.random.random_sample() * x.shape[0])
    x = x.iloc[randomId, :]
    y = y.iloc[randomId]
    gradient_arr = np.zeros(x.shape[0])
    gradient_beta = 0
    err = y - predict(alpha, beta, x)
    gradient_arr += err * x
    gradient_arr -= alpha * L2_lambda
    gradient_beta += err

    # arr 是 alpha vector的梯度vec,  alpha0 是 arr[0]
    gradient_arr = gradient_arr * 2
    gradient_beta = gradient_beta * 2

    alpha += np.reshape(gradient_arr, alpha.shape) * learn_rate
    beta += gradient_beta * learn_rate

    return alpha, beta
```

In [9]:

```
def train_model(x, y, learn_rate, loop_times, method):
    # random init alpha, beta
```

```
    alpha = np.random.random_sample(x.shape[1])
    beta = np.random.random_sample()
    mean_s_err = 0
    if method == "SGD":
        for i in range(loop_times):
            alpha, beta = gradient_descent_random(x, y, alpha, beta, learn_rate)
        return alpha, beta
    elif method == "SGD_L2":
        for i in range(loop_times):# 试试L2_lambda设置为3e-3 or 0.001 or 0.01
            alpha, beta = gradient_descent_random_L2(x, y, alpha, beta, learn_rate, 1)
        return alpha, beta
    else:
        for i in range(loop_times):
            alpha, beta, mean_s_err = gradient_descent(x, y, alpha, beta, learn_rate)
        return alpha, beta, mean_s_err
```

In [16]:

```
'''
flag = ["quality"]
x=data.loc[:, ~data.columns.isin(flag)].copy()
x = Normalization_fun(x)
y= data[flag].copy()
skf =StratifiedShuffleSplit(n_splits=5,test_size=0.25,train_size=0.75,random_state=10)
for train_index, test_index in skf.split(x, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    x_train, x_test = x.iloc[list(train_index)], x.iloc[list(test_index)]
    y_train, y_test = y.iloc[list(train_index)], y.iloc[list(test_index)]
'''
```

```
TRAIN: [4790 1658 4645 ... 4493 3998 4195] TEST: [1759 3737 2576 ... 4296 1741  193]
TRAIN: [ 825 2555 3436 ... 2779 4890 1962] TEST: [1727 3214 1606 ... 2251 1178 4210]
TRAIN: [1771 4695 2817 ... 2973 2595 4883] TEST: [2644 3421  959 ...  466  109 3181]
TRAIN: [1004 2968 1633 ... 4248  762 3828] TEST: [1173  594  755 ... 1110  687 2997]
TRAIN: [4194 2375 4593 ...  265 3832  413] TEST: [2711  780 2042 ... 4050 2915 2497]
```

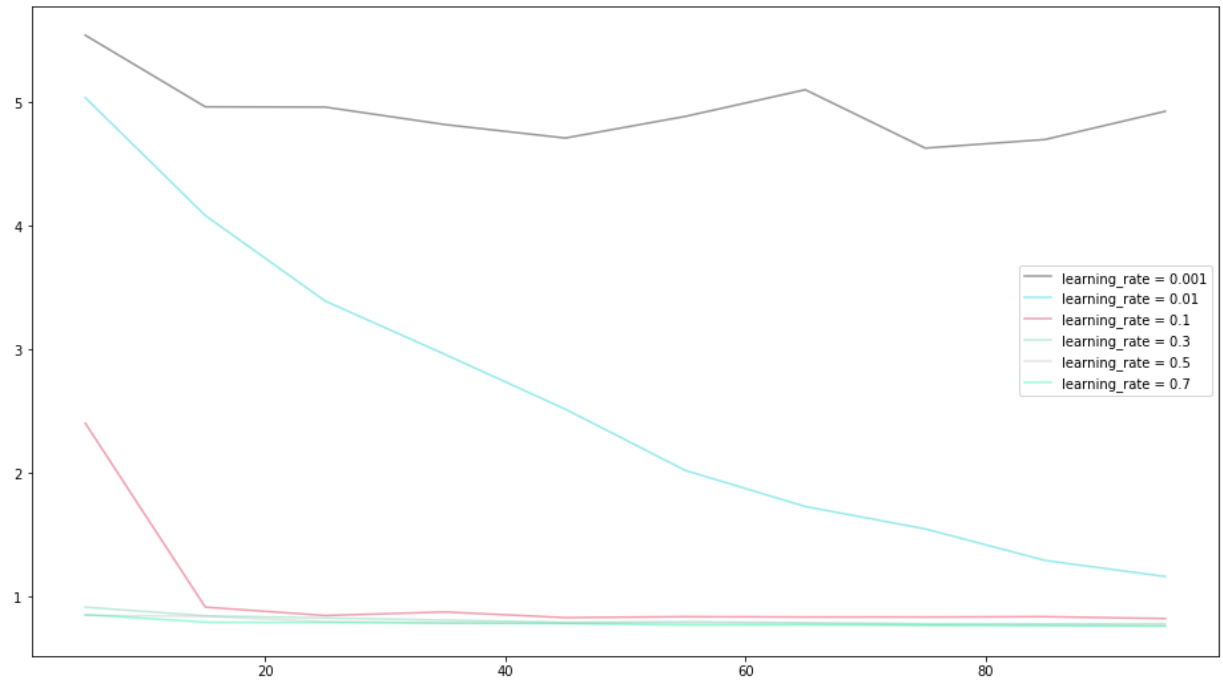
In [10]:

```
# BGD批量梯度下降
learn_rate = [0.001, 0.01, 0.1, 0.3, 0.5, 0.7]
mean_s_err_vec_dict = {}
x_loop = [i for i in range(5, 100, 10)]  # 迭代次数
MSE = []

X_train, Y_train, X_test, Y_test = StratifiedSampling(df)
def function(rate):
    mean_s_err = 0
    mean_s_err_vec = []
    for loop in x_loop:
        alpha, beta, mean_s_err = train_model(X_train, Y_train, rate, loop, "BGD")
        mean_s_err_vec.append(np.sqrt(mean_s_err))
    return mean_s_err_vec

for rate in learn_rate:
    mean_s_err_vec_dict[f'learning_rate = {rate}'] = function(rate)

colors = ['#000000', '#00CED1', '#DC143C', '#66CDAA', '#BEBEBE', '#00FA9A','#FF00FF']
points = np.pi * 0.6 ** 2

fig, ax = plt.subplots(figsize=(16, 9))
i = 0
for k, v in mean_s_err_vec_dict.items():
    print(*v,"\n")
    # plt.scatter(x_loop, v, s=points, alpha=0.4, c=colors[i], label=k)
    print(k,"\t",*v,"\n")
    ax.plot(x_loop, v, alpha=0.4, c=colors[i], label=k)
```

```
    i += 1
plt.legend()
# plt.show()
```

```
5.538509677639809 4.959629287255401 4.957454864374464 4.815874422475323 4.707199203791
742 4.882174141712975 5.098106122324515 4.625782144377061 4.695478445510453 4.92352580
1930998

learning_rate = 0.001    5.538509677639809 4.959629287255401 4.957454864374464 4.81587
4422475323 4.707199203791742 4.882174141712975 5.098106122324515 4.625782144377061 4.6
95478445510453 4.923525801930998

5.033526218723557 4.080575683454252 3.3888003045172947 2.956657858304112 2.51501437783
16965 2.0196230177692613 1.7274822452473844 1.546363545617855 1.2910610381462 1.161056
6608562238

learning_rate = 0.01     5.033526218723557 4.080575683454252 3.3888003045172947 2.9566
57858304112 2.5150143778316965 2.0196230177692613 1.7274822452473844 1.546363545617855
1.2910610381462 1.1610566608562238

2.4004189242118996 0.9134518572540091 0.8458976553165981 0.8745763413069756 0.82887802
71568708 0.8366335174806813 0.8340231564782321 0.8336448151765826 0.8366371334978718
0.8212142825823594

learning_rate = 0.1      2.4004189242118996 0.9134518572540091 0.8458976553165981 0.87
45763413069756 0.8288780271568708 0.8366335174806813 0.8340231564782321 0.833644815176
5826 0.8366371334978718 0.8212142825823594

0.9131691034324506 0.8436307268103175 0.8259163265244526 0.808852286919524 0.790484185
7320296 0.7954555369430618 0.7878931569766764 0.7807084391102952 0.7784046726819751 0.
7797133813438427

learning_rate = 0.3      0.9131691034324506 0.8436307268103175 0.8259163265244526 0.80
8852286919524 0.7904841857320296 0.7954555369430618 0.7878931569766764 0.7807084391102
952 0.7784046726819751 0.7797133813438427

0.8469362808007506 0.8391148659422175 0.7967979118934123 0.7898673064773013 0.78493289
50243754 0.7832029458442233 0.7802202997237223 0.7691556071147436 0.7696281351487482
0.7657150530056865

learning_rate = 0.5      0.8469362808007506 0.8391148659422175 0.7967979118934123 0.78
98673064773013 0.7849328950243754 0.7832029458442233 0.7802202997237223 0.769155607114
7436 0.7696281351487482 0.7657150530056865

0.8532518612247831 0.7922832516608183 0.7899228190251995 0.7826863782710506 0.78068480
20833903 0.7684015905288626 0.7695763101228191 0.767641971923047 0.7632506741042999 0.
7587902751464046

learning_rate = 0.7      0.8532518612247831 0.7922832516608183 0.7899228190251995 0.78
26863782710506 0.7806848020833903 0.7684015905288626 0.7695763101228191 0.767641971923
047 0.7632506741042999 0.7587902751464046
```

Out[10]: <matplotlib.legend.Legend at 0x1af0e7c52e0>



结论

通过实验数据可知，学习率过高或过低时都会影响模型准确率。学习率过高会导致过拟合，过低会导致欠拟合。而且，随着loop次数的增加，MSE在稳步下降。由图可知，对于批量梯度下降来说，最佳学习率大约为0.3~0.7之间，它们的MSE的差异随着loop次数的增加逐渐减小，以致基本相同。

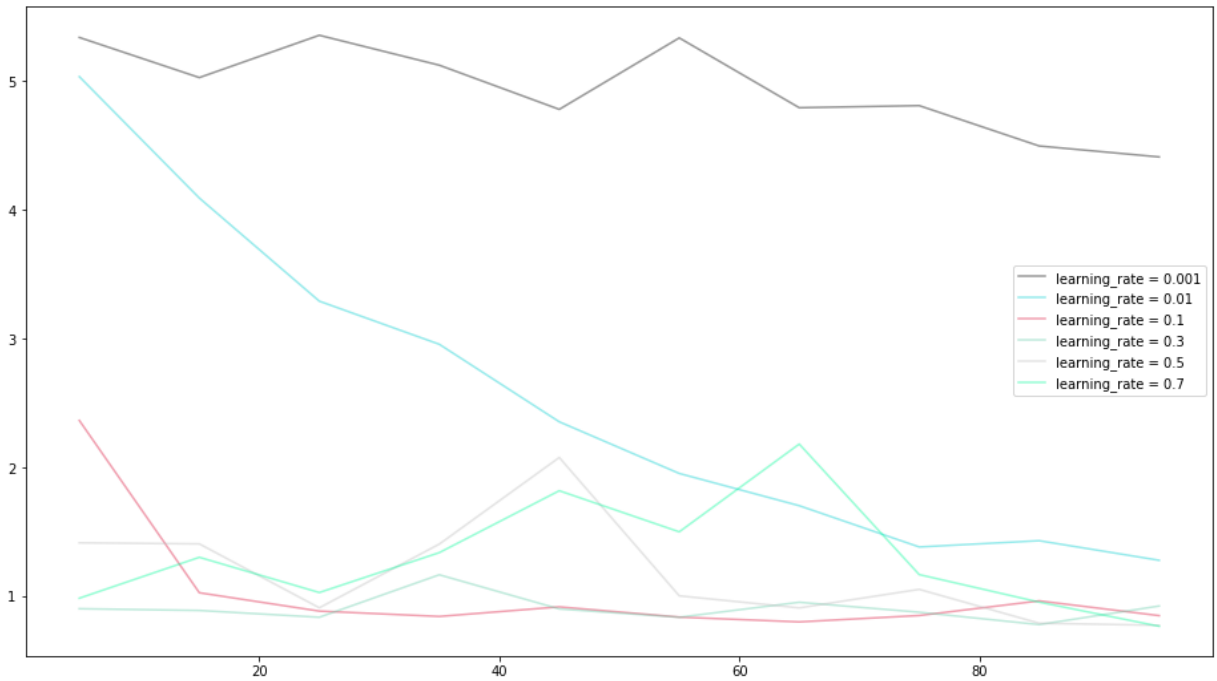
In [11]:

```
# SGD随机梯度下降
learn_rate = [0.001, 0.01, 0.1, 0.3, 0.5, 0.7]
mean_s_err_vec_dict = {}
x_loop = [i for i in range(5, 100, 10)]  # 迭代次数

X_train, Y_train, X_test, Y_test = StratifiedSampling(df)
def function(rate):
    mean_s_err = 0
    mean_s_err_vec = []
    for loop in x_loop:
        alpha, beta = train_model(X_train, Y_train, rate, loop, "SGD")
        for i in range(X_test.shape[0]):
            err = Y_test.iloc[i] - predict(alpha, beta, X_test.iloc[i, :])
            mean_s_err += err ** 2
        mean_s_err /= X_test.shape[0]
        mean_s_err_vec.append(np.sqrt(mean_s_err))
    return mean_s_err_vec

for rate in learn_rate:
    mean_s_err_vec_dict[f'learning_rate = {rate}'] = function(rate)

colors = ['#000000', '#00CED1', '#DC143C', '#66CDAA', '#BEBEBE', '#00FA9A','#FF00FF']
points = np.pi * 0.6 ** 2

fig, ax = plt.subplots(figsize=(16, 9))
i = 0
for k, v in mean_s_err_vec_dict.items():
    # plt.scatter(x_loop, v, s=points, alpha=0.4, c=colors[i], label=k)
    print(k,"\t",*v,"\n")
    ax.plot(x_loop, v, alpha=0.4, c=colors[i], label=k)
    i += 1
```

```
plt.legend()
# plt.show()
```

```
learning_rate = 0.001     5.338203211505956 5.0265912807652935 5.355223609767796 5.1228
42419743947 4.779781934826513 5.334725036265477 4.792622392594266 4.808853150835543 4.
495431968509427 4.410351321608804

learning_rate = 0.01      5.033596625917952 4.09101513250736 3.2903488806110794 2.95553
04143355894 2.3535086591809975 1.9523834897521122 1.7020294398971243 1.381479001894816
9 1.4300773138087373 1.2780520525522252

learning_rate = 0.1       2.3636219239123375 1.025995172307876 0.8834647071527226 0.842
0556590230688 0.9162468525630373 0.8367372258121193 0.7996594751454207 0.8491529373690
629 0.9628376034865614 0.8482775725239529

learning_rate = 0.3       0.9018863907454857 0.8883159646138916 0.8357219604180539 1.16
5491408776547 0.9001775443749306 0.835717516523939 0.9520796791620731 0.87472187315948
45 0.7785741676241292 0.9230583406441516

learning_rate = 0.5       1.4135922249992836 1.405501269769815 0.9091415532840411 1.403
7562887034716 2.0763483459327023 1.0020070153583296 0.9084587527762953 1.0523261452952
544 0.7891662371599315 0.7733603193910354

learning_rate = 0.7       0.983570005226202 1.3013631463393844 1.027602603091514 1.3375
525394919603 1.8182321719082668 1.4989996361535511 2.18072279225685 1.1669798262827324
0.9531636816705917 0.7655400957961416
```

Out[11]: <matplotlib.legend.Legend at 0x1af41b3df10>



结论

对于随机梯度下降来说，最佳学习率在0.1~0.3之间，且随着loop次数的增加MSE差异逐渐变小。

中级要求

探究回归模型在机器学习和统计学上的差异。

回归模型在机器学习和统计学上的差异

机器学习和统计学之间的主要区别在于它们的目的：**机器学习模型旨在使最准确的预测成为可能；统计模型被设计用于推断变量之间的关系。**

1. 统计学模型与机器学习在线性回归上的差异

线性回归是一种统计方法，我们训练线性回归量并获得与统计回归模型相同的结果，旨在最小化数据点之间的平方误差。

在机器学习中，我们做了"训练"模型的事情，其中涉及使用数据的一个子集。我们不知道模型将如何执行，直到在训练期间能够"测试"出此数据不存在的、被称为测试集的其他数据。在这种情况下，机器学习的目的是在测试集上获得最佳性能。

对于统计模型，我们只要找出可以最小化所有数据的均方误差（假设数据是一个线性回归量，加上一些随机噪声，本质上通常是高斯噪声），无需训练，也无需测试。

一般来说，特别是在研究中，模型的要点是表征数据与结果变量之间的关系，而不是对未来数据进行预测。我们将此过程称为统计推断，而不是预测。但我们仍然可以使用此模型进行预测，但评估模型的方式不涉及测试集，而是涉及评估模型参数的重要性和稳健性。

（受监督的）机器学习的目的是获得可以进行可重复预测的模型。我们通常不关心模型是否可解释，机器学习只看重结果。而统计建模更多的是发现变量之间的关系和这些关系的重要性，同时也适合预测。

2. 机器学习是基于统计学的

机器学习建立在统计框架之上。统计学的理论是机器学习的统计基础。

统计学与机器学习之间的主要区别在于统计学仅基于概率空间。从集合论中推导出整个统计数据，它讨论了我们如何将数字组合成类别，称为集合，然后对此集合强加一个度量，以确保所有这些的总和值为1，我们称之为概率空间。

除了这些集合和度量的概念之外，统计数据不对宇宙做任何其他假设。这就是为什么当我们用非常严格的数学术语指定概率空间时，我们指定了3个东西。

概率空间，我们这样表示，（Ω，F，P）由三部分组成：

- 样本空间Ω，它是所有可能结果的集合
- 一组事件F，其中每个事件是包含零个或多个结果的集合
- 为事件分配概率P; 也就是说，从事件到概率的函数

根据监督学习的统计学习理论，一组数据，我们将其表示为$S=\{(x_i，y_i)\}$。这是一个有n个数据点的数据集，每个数据点由我们称之为功能的其他一些值描述，这些值由x提供，并且这些特征由某个函数映射以给出值y。

假如说我们已经有了这些数据，我们的目标是找到将x值映射到y值的函数。可以描述此映射的所有可能函数的集合，称为假设空间。

要找到这个函数，我们必须让算法"学会"一些方法来找出解决问题的最佳方法，这个过程由损失函数实现。因此，对于我们所拥有的每个假设（建议函数），需要通过查看其对所有数据的预期风险值来评估该函数的执行情况。

预期风险基本上是损失函数乘以数据概率分布的总和。如果我们知道映射的联合概率分布，就很容易找到最佳函数。然而，这通常是未知的，因此我们最好的选择是猜测，然后凭经验确定损失函数是否更好。我们称之为经验风险。

然后，我们可以比较不同的函数，并寻找给出最小预期风险的假设，即假设给出数据上所有假设的最小值（称为下限）。

然而，该算法具有作弊的倾向，可以通过过度拟合数据来最小化其损失函数。这就是为什么在学习基于训练集数据的函数之后，该函数需要在测试数据集上进行验证，验证用的数据数据不会出现在训练集中。

显然，这不是统计学看重的点，因为统计学并不需要最小化经验风险。选择最小化经验风险的函数的学习算法称为经验风险最小化。

高级要求

编程实现岭回归算法，求解训练样本的岭回归模型，平均训练误差和平均测试误差（解析法、批量梯度下降法和随机梯度下降法均可）。

In [10]:
```
# 岭回归 这个函数前面写了，这里不需要跑
def gradient_descent_random_L2(x, y, alpha, beta, learn_rate, L2_lambda):
    L2_lambda = 0.01
    randomId = int(np.random.random_sample() * x.shape[0])
    x = x.iloc[randomId, :]
    y = y.iloc[randomId]
    gradient_arr = np.zeros(x.shape[0])
    gradient_beta = 0
    err = y - predict(alpha, beta, x)
    gradient_arr += err * x
    gradient_arr -= alpha * L2_lambda# 加上L2 penalty
    gradient_beta += err

    # arr 是 alpha vector的梯度vec,  alpha0 是 arr[0]
    gradient_arr = gradient_arr * 2
    gradient_beta = gradient_beta * 2

    alpha += np.reshape(gradient_arr, alpha.shape) * learn_rate
    beta += gradient_beta * learn_rate

    return alpha, beta
```

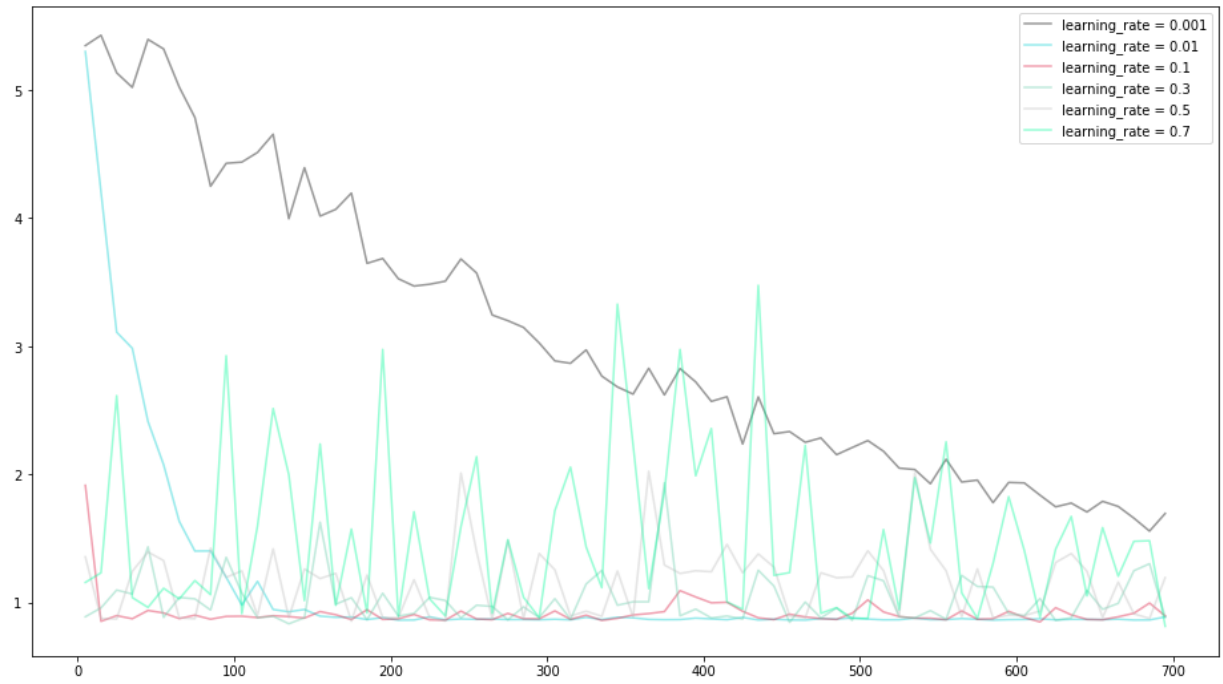
In [13]:
```
# 岭回归 随机梯度下降 L2_lambda = 1
learn_rate = [0.001, 0.01, 0.1, 0.3, 0.5, 0.7]
mean_s_err_vec_dict = {}
x_loop = [i for i in range(5, 700, 10)]  # 迭代次数

X_train, Y_train, X_test, Y_test = StratifiedSampling(df)
def function(rate):
    mean_s_err = 0
    mean_s_err_vec = []
    for loop in x_loop:
        alpha, beta = train_model(X_train, Y_train, rate, loop, "SGD_L2")
        for i in range(X_test.shape[0]):
            err = Y_test.iloc[i] - predict(alpha, beta, X_test.iloc[i, :])
            mean_s_err += err ** 2
        mean_s_err /= X_test.shape[0]
        mean_s_err_vec.append(np.sqrt(mean_s_err))
    return mean_s_err_vec

for rate in learn_rate:
    mean_s_err_vec_dict[f'learning_rate = {rate}'] = function(rate)
```

```
colors = ['#000000', '#00CED1', '#DC143C', '#66CDAA', '#BEBEBE', '#00FA9A','#FF00FF']
points = np.pi * 0.6 ** 2

fig, ax = plt.subplots(figsize=(16, 9))
i = 0
for k, v in mean_s_err_vec_dict.items():
    # plt.scatter(x_loop, v, s=points, alpha=0.4, c=colors[i], label=k)
    print(k,"\t",*v,"\n")
    ax.plot(x_loop, v, alpha=0.4, c=colors[i], label=k)
    i += 1
plt.legend()
# plt.show()
```

```
learning_rate = 0.001    5.347935317429053 5.428510573071651 5.1355604372759265 5.0213
23222609492 5.396573539225938 5.321552503238591 5.02396503664282 4.786568944768707 4.2
49845478377553 4.428925330153538 4.438365537106184 4.513700794539037 4.655898679567757
3.9961301684076447 4.394958601431716 4.016794745340161 4.069404120561908 4.19632179003
5234 3.647767032965497 3.685861169005132 3.527672635439909 3.470439356073356 3.4848722
473461597 3.508412004686848 3.6820917272962332 3.5722371352881237 3.2443776773613666
3.200175330514277 3.147559322151068 3.0264213045317905 2.8862457580495424 2.8674459224
83717 2.972049987056205 2.76765898025732 2.6830127723191652 2.626865882140729 2.828608
4396588866 2.620374723348787 2.8263575369382004 2.72215700900782 2.57013138825014 2.60
63582806871874 2.2372434530566343 2.6052036503547678 2.318027138361738 2.3345172582711
31 2.2507471269602872 2.284924657942203 2.154060137590521 2.2096987154399526 2.2642957
286278724 2.180805494350848 2.0491823864543486 2.0380583805346775 1.9270504254822365
2.118262643068237 1.9404940517475007 1.9558001679917472 1.7794226557657293 1.937429030
7603663 1.9334130515200512 1.8382859312623256 1.7469481379081655 1.7761728465246096 1.
706541056845366 1.7899476236227703 1.7498504297189035 1.6587120718438024 1.55741966848
39434 1.695027707021414

learning_rate = 0.01     5.301292548858653 4.205623855256422 3.1107185788653555 2.9846
62145266376 2.4142847580059086 2.075981734814505 1.6335120345384861 1.401462727329581
1.4023385529262684 1.1965489271341336 0.9797508023143437 1.1651992612802011 0.94533975
93896277 0.9276300573929238 0.945311499208238 0.8937064032285869 0.8844983317837737 0.
8868458617473913 0.8670416692053182 0.8821370170453758 0.862133482390442 0.86309105317
77642 0.8885155862779784 0.8640521162236041 0.8716199901889009 0.8686032708992611 0.87
14108763541011 0.8664634398923621 0.8670266790576396 0.864906315639387 0.8694979873473
844 0.8647213459472374 0.8839761214603467 0.8673847110374981 0.8850063600616945 0.8805
03404770088 0.8681096697468315 0.8658877755015668 0.8668767104064858 0.878133096088942
1 0.8718570424320358 0.8657610753769034 0.883718156280188 0.8637284640828937 0.8695280
404465991 0.8637022001218564 0.8626648466312017 0.8746155845062251 0.8704025696887174
0.8796456104715171 0.870949602105145 0.8647247609890545 0.8654470898728125 0.877778440
7643695 0.8661789883698781 0.8668753118329318 0.8743858739593235 0.8689107493453074 0.
8625848920071831 0.8662630734668165 0.8675395708646182 0.8727747345366546 0.8648732911
002929 0.8684476103288604 0.8681195516377014 0.8664414959171479 0.8697879981119254 0.8
62935559158636 0.8637353336288629 0.8873531617348345

learning_rate = 0.1      1.914276296408305 0.8531948881372736 0.897607762268592 0.8732
418322474911 0.9378445377353778 0.9185176775772735 0.8745182184559244 0.90124973519120
78 0.8697404193708427 0.8916733199715587 0.892915083254199 0.8826008222892361 0.896514
2814468859 0.8903202653730828 0.8785111268111979 0.9291337514255202 0.9045197986510402
0.8668122252067385 0.9436992614414513 0.867864132821216 0.8725774963701383 0.906459870
1641728 0.8646019082190018 0.8617869719786442 0.9328981665899515 0.8715328295192487 0.
8661756784737714 0.9148940792363381 0.8732868212974334 0.8723638116623834 0.9341982994
039291 0.8685806146578718 0.9009956197680631 0.8602386375611172 0.876482932195021 0.90
2754121450202 0.9136611117595279 0.9289186536274212 1.0925551948929708 1.0431302524697
494 0.9975548878065861 1.0018871597937893 0.9313509130615502 0.8803919207029522 0.8666
38397982978 0.9085872719446773 0.8872657345172758 0.8737199477428098 0.868024237047356
8 0.9161167038361541 1.0202028140841193 0.9263053443970204 0.8905882413488703 0.877799
9053710955 0.8771561955224044 0.8653188794615 0.9343719858587647 0.8688856654577848 0.
8735403647142995 0.9313109926787169 0.8829376246622521 0.8481759439584707 0.9593081536
478132 0.903398932883827 0.868121401593591 0.8652573675251842 0.8884049874686158 0.916
0008077774985 0.9957730206123793 0.8951532358640913

learning_rate = 0.3      0.8887603228159751 0.9582001263738742 1.0973007973783595 1.06
64888350428872 1.4340226062494619 0.8835805177682585 1.0397147463272913 1.028411717361
644 0.9414989607711164 1.3527263864279926 1.074044672000898 0.8803968323181847 0.89247
15488548025 0.8339515437144714 0.8806606442172382 1.6267939126053004 0.986860144175253
```

```
8 1.0372103078302337 0.8656584600782876 1.07228246058061 0.8945971804314428 0.91440263
84261895 1.040137604356959 1.0160458870794966 0.8799660617965791 0.9789988734143811 0.
9690084334318524 0.8644967918896743 0.9660826857855581 0.8823030001698273 1.0296476678
582391 0.8691836887998479 1.1444735602271987 1.250556464926649 0.978719810372441 1.005
6026204531068 1.0057050018249347 1.9340865492864627 0.8960004576308862 0.9484653649014
225 0.8791929701783094 0.8945606397601412 0.8710706883399207 1.2520969723832907 1.1284
974130346537 0.8445628537134748 1.003597056570029 0.8793748313927384 0.956498084806224
9 0.8614105309465191 1.209740305474523 1.1699827405633212 0.889210990569023 0.87931403
22075469 0.9365891703260409 0.8694982028805311 1.2115589406272043 1.1250881586217458
1.1202971189178892 0.915029549234119 0.8966442457847676 1.0306936139248117 0.859886271
3049243 0.880374861437991 1.095187186314646 0.9470075535703535 0.9934234104654226 1.24
80764726178548 1.3027892583565945 0.885625540524444

learning_rate = 0.5      1.3560074539153093 0.8756282626697155 0.8712569529399552 1.24
32993492924218 1.3944334198777721 1.3274318669891954 0.8799674092528954 0.873129607285
721 1.4241473760943624 1.1967912832164 1.2466526120458448 0.8987734555897462 1.4184812
45898516 0.882583777377949 1.2615776111514125 1.1868039669498043 1.2278257771580352 0.
8547633573735763 1.2137124911213133 0.8840042341329064 0.8809567989954812 1.1777079616
669954 0.8820024111442101 0.855690145217539 2.0102911054171333 1.3964855254014144 0.86
92083285255883 1.4833262773341787 0.8759288613305192 1.3825802564646807 1.255382272053
1413 0.8944602463729336 0.9318136992791408 0.8924733769087128 1.2458442820430158 0.885
4124289165974 2.0239817950501613 1.2927992804776243 1.2267625843256098 1.2478325558087
158 1.2402039840978707 1.4540676029758794 1.231925544802184 1.379271200792837 1.276200
4095035124 0.8955467998903884 0.879606592388 1.2324159619930108 1.193264148997298 1.20
00318031521289 1.4037576245848105 1.2488617947449756 0.8915926881376309 2.012946742459
481 1.4149548411777015 1.247902779273065 0.8893319837432481 1.2627507667951872 0.88017
66453409516 0.9000921985255671 0.8976296219643075 0.9332481838288049 1.312228225423971
2 1.3820360205085276 1.2440709371388576 0.8823647293299878 1.1589367532972856 0.911639
1930486318 0.8751109569794654 1.1937370068568227

learning_rate = 0.7      1.1566924493388449 1.2319101034499322 2.614595929566226 1.038
4317336729871 0.9617687623743065 1.1108363238102832 1.027872469222275 1.17128272496287
74 1.061308533544375 2.926511639519803 0.9131139352796542 1.5971523966311179 2.5145272
90690777 1.9984343533610371 1.0147133113833409 2.2378943377340685 0.9779471863582472
1.5732652904463482 0.9177825565865388 2.97575593262282 0.8934205410393251 1.7090040644
21701 1.0285840510211834 0.8952183402323566 1.5916216665156724 2.1383443500515877 0.91
61874774701310 51 1.4914909149376097 1.0402761797634288 0.8820480491648972 1.716770194957
8373 2.056771954249933 1.433861943852532 1.1159797457496319 3.329328080485151 2.230375
120049681 1.1084891999314297 1.8619219124698512 2.975981991045689 1.989345188420575 2.
358639200830521 1.0040149624211268 0.9481377789626376 3.47634116855306 1.2116672975855
08 1.2334698906679546 2.2268317662158377 0.915272472565977 0.9599851920839476 0.876163
5384407097 0.8761324402255702 1.5698613533832824 0.9425417047518044 1.9738315230686794
1.4645102000405563 2.254367894486481 1.0747485036614965 0.8759690767636503 1.304626548
164312 1.8265410991405175 1.4052142286305087 0.8841763705144459 1.4167564555157266 1.6
718390086982953 1.0512800885129427 1.584720744719547 1.2055900327183384 1.477547853163
8 1.4822211690449623 0.8143134330269802
```

Out[13]: <matplotlib.legend.Legend at 0x1af41b3dee0>



In [11]:

```
# 岭回归 随机梯度下降 L2_lambda = 0.01
learn_rate = [0.001, 0.01, 0.1, 0.3, 0.5, 0.7]
mean_s_err_vec_dict = {}
x_loop = [i for i in range(5, 700, 10)]  # 迭代次数

X_train, Y_train, X_test, Y_test = StratifiedSampling(df)
def function(rate):
    mean_s_err = 0
    mean_s_err_vec = []
    for loop in x_loop:
        alpha, beta = train_model(X_train, Y_train, rate, loop, "SGD_L2")
        for i in range(X_test.shape[0]):
            err = Y_test.iloc[i] - predict(alpha, beta, X_test.iloc[i, :])
            mean_s_err += err ** 2
        mean_s_err /= X_test.shape[0]
        mean_s_err_vec.append(np.sqrt(mean_s_err))
    return mean_s_err_vec

for rate in learn_rate:
    mean_s_err_vec_dict[f'learning_rate = {rate}'] = function(rate)

colors = ['#000000', '#00CED1', '#DC143C', '#66CDAA', '#BEBEBE', '#00FA9A','#FF00FF']
points = np.pi * 0.6 ** 2

fig, ax = plt.subplots(figsize=(16, 9))
i = 0
for k, v in mean_s_err_vec_dict.items():
    # plt.scatter(x_loop, v, s=points, alpha=0.4, c=colors[i], label=k)
    print(k,"\t",*v,"\n")
    ax.plot(x_loop, v, alpha=0.4, c=colors[i], label=k)
    i += 1
plt.legend()
# plt.show()
```

```
learning_rate = 0.001    5.742970673613277 5.599493247548835 5.463107896216411 4.66324
4596857244 5.194664966173732 4.683787775239729 4.985276271441249 5.008163843511597 4.7
96738767327236 4.284303144791928 4.283538794447952 4.247381125647713 4.213678861463233
3.9166538624265455 3.8803705489114653 3.9099211313790323 3.763255976503751 4.060323551
471607 4.09346967972256 3.728596147491654 3.9515384693956954 3.2933547099627387 3.7099
243811807927 3.379726978058118 3.526616551506736 3.555061380598461 3.094819951614923
3.160469105516166 3.060929190008779 3.2716662196098243 3.218825580761119 2.85848319666
```

1214 3.139983994819742 2.8828038481511964 2.77826984988479 2.580704822122552 2.6440798
17502028 2.8165256105006207 2.5804470913472115 2.679919882787074 2.3906315360721604 2.
621664313017973 2.311958403497999 2.2928029341272906 2.177628699022051 2.3452970682661
336 2.3511574591942748 2.2790723133450532 2.3631473357204666 2.1378992572156656 2.2695
828717797677 2.1413416663149225 2.2009126742659473 1.8690714046290184 2.09140967811607
1.9813692005190853 1.8658552805375748 1.7781461615358323 1.8580551253515376 2.00574859
3196005 1.9119039727851914 1.8985526550941614 1.6636094685861307 1.7158859972229499 1.
778225393442155 1.6185736170983744 1.7666637490060302 1.5056904805305487 1.73250094836
57647 1.6786234927625552

learning_rate = 0.01 5.031544544772799 4.237972722646492 3.635834847953212 3.03721
4684777021 2.214576900173948 1.8449038350923594 1.8358195120207927 1.5644447584006769
1.2004451373589913 1.1677782986022864 1.166336722855075 0.9601635242768153 0.968331766
167306 0.9048438553656781 0.8965216872048418 0.8893961150506468 0.9214877690071496 0.8
903789130063196 0.8867991293714355 0.8568197798996958 0.8728276275349439 0.92565739719
66642 0.8476506905997424 0.852621407464143 0.8921075732577343 0.8561728196001424 0.828
054857720691 0.8845408502180386 0.8465107587133388 0.8715690609899059 0.85120662080141
96 0.812684619026106 0.8961962153918062 0.8429039098292557 0.8501344188844967 0.838618
0864715261 0.8399150512027991 0.842328884188906 0.8304523272112537 0.8385857813348038
0.8559788428155167 0.8278211776608626 0.8455878605162805 0.8474162884310205 0.82716417
70498456 0.8385418817394049 0.8355663738199777 0.8193210931483048 0.8335311364432986
0.8277906139327369 0.8278249921427896 0.8351866826035987 0.8288488237278662 0.83054754
88675906 0.8446478022853146 0.8430201991567878 0.8099746609371096 0.8121902917142492
0.8193235079801434 0.8280756998456745 0.8201270689875543 0.8300874337746353 0.81164764
10942938 0.8233798440108109 0.8132124112208959 0.8292240845155254 0.8065337719636587
0.8336309943419292 0.821982458080017 0.8362826565770936

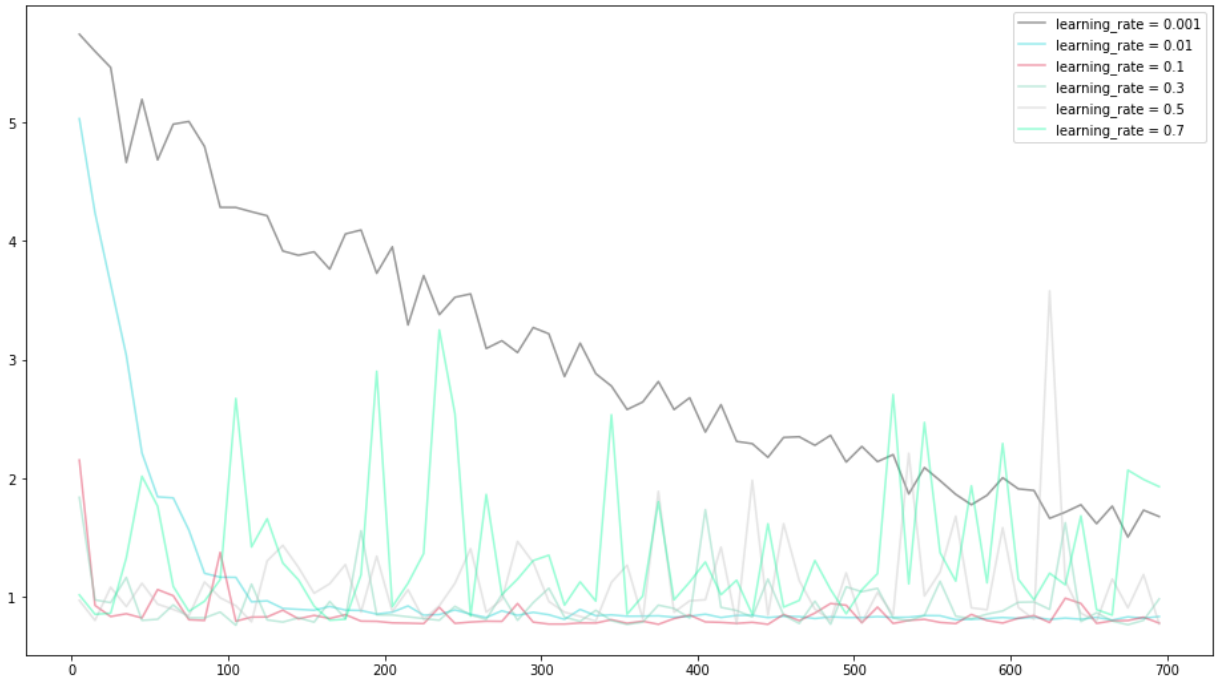
learning_rate = 0.1 2.154776754380073 0.9299468561216607 0.8359974547592678 0.859
7120773612822 0.8239037982002264 1.0633970088025237 1.011477908085717 0.80925721745053
51 0.8032425276638537 1.3773455461102384 0.7967608557392012 0.83117892487526 0.8327676
607498784 0.889724969362864 0.8167196332158311 0.8458015855104156 0.8199775968651185
0.8520412690148306 0.7974994359546357 0.7956388064193107 0.7827625756523892 0.78013292
19857135 0.77653826952384 0.9141453157625391 0.7783323316051243 0.7899148984858835 0.7
96755522904699 0.7954665635640802 0.9454831048509926 0.7891409938090872 0.772633727868
1308 0.7726324384878593 0.7820251245289234 0.7813911902186391 0.8095808391800831 0.779
4785841563374 0.7968429314515424 0.7705976345802176 0.8188978626624991 0.8483719366283
161 0.7909072518773784 0.7871712250555852 0.778395190315811 6 0.7875191305247443 0.7704
982737290741 0.8534840268547788 0.8014518538062052 0.8652863286793014 0.94598581519199
98 0.9314246863805091 0.7838455873951728 0.9157135798405746 0.7778926984611308 0.80340
35827156556 0.8123216512283975 0.7867623968309607 0.7767182476586894 0.853790037602583
8 0.8026316708972321 0.7811112429265124 0.8219330066675992 0.8447803361103743 0.785705
9720424877 0.9906067345135049 0.9460304100115913 0.7784402246999697 0.8001783702436298
0.8024979225209293 0.8303859384302017 0.7804536124303134

learning_rate = 0.3 1.8402164788412476 0.9764445037980575 0.9533965367144881 1.16
60660210212190268 0.8039936229235759 0.8130052339280426 0.9344489662072644 0.825767656033
6557 0.8245361389049771 0.8740374654548843 0.7623730157461023 1.1091968104159247 0.808
0075415963369 0.7894190834131359 0.8254139628924053 0.7888676548789766 0.9628444209579
07 0.8196114970652143 1.5596677105646342 0.8502373197274178 0.8491455766739786 0.83406
67942765767 0.8189415704985005 0.8049494490435884 0.921582260958 8487 0.849242696398392
1 0.8118313604836818 1.0216018343954096 0.8048805630906826 0.9531215083986967 1.075106
6965586662 0.8306032120881804 0.7956536098755821 0.8883399117979218 0.8016091526806217
0.7696405740860294 0.7892545275472778 0.9322581572825359 0.903809344766209 0.822938249
9216019 1.7367587719789226 0.913683996969525 0.8866468716096709 0.8318670459988 1.1519
532514854118 0.8481674419923269 0.7777010510881912 0.9657660556376768 0.77126905760203
29 1.0862765099633154 1.0452693099126198 1.0738267557833547 0.8231933541188123 0.80090
142786004 0.8459609793282852 1.1328235529665838 0.8395103657055245 0.8178520668275931
0.8563761328219319 0.8830968726531448 0.9561347075981448 0.9590382939355877 0.89667233
21598173 1.6271492058654553 0.7932766699658874 0.8683233726088064 0.7982731772951955
0.7671477810865964 0.8040384582655982 0.9846076395969785

learning_rate = 0.5 0.9722524053070071 0.8032340447244632 1.0831052920529471 0.92
01641136478247 1.1162643429474894 0.9408976569051809 0.8962435649623353 0.843980603578
651 1.1274518990965796 0.9937101692495234 0.9274966604498823 0.7918636378779489 1.3045
631858408244 1.4360869796796045 1.250466537370735 1.0323254215402018 1.112754915772083
3 1.274402455698444 0.8744255475044302 1.3447439197286286 0.8777704207647548 1.0619997
897353686 0.812453524139835 0.9265469451750507 1.116115422790437 1.4076613568278133 0.
8727343896313499 0.9799013666942249 1.4687517667607943 1.291778394605689 0.96347309827
36782 0.8731113902351609 0.8367574683169771 0.7979072151708578 1.1243263516114421 1.26

```
8748855568837 0.817366292047152 1.8903635180495089 0.8722299036975366 0.96852091206813
18 0.9776061160679897 1.4207581175269177 0.7731318866624945 1.9838286571769543 0.84656
60190336047 1.6185412739242628 1.139271422099827 0.898535970088876 0.8045051199191643
1.203824887167168 0.7842184855381175 1.0391447051929308 0.8507858720194327 2.211352520
89211 1.0093379952340737 1.2115383352831477 1.6812574942981022 0.9095113845544714 0.89
31466462039962 1.585087867316194 0.912516386704665 0.8127244176591351 3.58042693055418
8 1.1366773586637673 0.8639824615363181 0.7981044876025822 1.1525156122832099 0.909576
7513413738 1.1901171366960712 0.7738072801649816

learning_rate = 0.7      1.0181774336670004 0.8552149836583761 0.8641222237314161 1.33
08128445851555 2.0183740498148994 1.7664058227368309 1.089784831231622 0.8796742640846
259 0.9657606810086204 1.1481292107785195 2.6739710845482554 1.4223853845967425 1.6606
04187720699 1.2873396833674933 1.1451407619675826 0.9242483055337066 0.804963059331303
1 0.8135716995806637 1.1881279344126663 2.9027648669779915 0.919856278528492 1.1167263
788748243 1.3665987317075927 3.251639320856576 2.544510056511413 0.8391948987467353 1.
8651512637278225 1.021259089658438 1.1518389126220132 1.3076095996623756 1.35297172320
43994 0.9320471375218551 1.1282372073375526 0.9660803133737674 2.5369130904515127 0.85
50184188771645 1.0092698593993714 1.8036607498559192 0.976919135173052 1.1269662892120
307 1.296211160537862 1.020161020680805 1.1421467236197278 0.8538334316889666 1.618023
7253809298 0.915479292893318 0.9718076330125183 1.3079733927182466 1.064525193333027
0.85912582065982 1.0616952058180058 1.1978452986932977 2.707044885962707 1.11141114941
89645 2.471902063975532 1.3746883800426357 1.1346523815781577 1.9376472465732864 1.120
001772602228 2.294553230120593 1.1503103423462604 0.9771918897379732 1.201578482780327
5 1.1033633942729708 1.6844774223597816 0.8941126772888034 0.8481510712196649 2.069787
4659281905 1.9913561645264308 1.9306028200432532
```

Out[11]: <matplotlib.legend.Legend at 0x24c29788820>



结论

由于loop次数设置为100时各个learning_rate的MSE都还不够稳定，因此将loop次数调高至700。由图可知，对于岭回归，最佳学习率为0.3。

In [ ]: