Artificial Intelligence - Week 1 Homework

Question 1

We start by including the Gaussian distribution used to draw the X

$$\begin{aligned} p(x,y) &= \sum_{i\in(1,2)} p(x,y,c(x)=i) \\ &= \sum_{i\in(1,2)} p(x,y \mid c(x)=i)\cdot p(c(x)=i) \end{aligned}$$

as y and x are independent, but both are dependent on $c(x)$, it can be written as follows.

$$p(x,y) = \sum_{i\in(1,2)} p(x \mid c(x)=i)\cdot p(y \mid c(x)=i)\cdot p(c(x)=i)$$

where

$$p(x \mid c(x)) = \begin{cases} \sim N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}9 & 0\\0 & 1\end{bmatrix}\right) & if\ c(x)=1 \\ \sim N\left(\begin{bmatrix}2.5\\0\end{bmatrix}, \begin{bmatrix}9 & 0\\0 & 1\end{bmatrix}\right) & if\ c(x)=2 \end{cases}$$

$$p(y=0 \mid c(x)) = \begin{cases} 0.8 & if\ c(x)=1 \\ 0.7 & if\ c(x)=2 \end{cases}$$

$$and \quad p(c(x)=1) = 0.8$$

Thus the distribution becomes

$$p(x,y) = p(x \mid c(x)=1)\cdot p(y \mid c(x)=1)\cdot p(c(x)=1) + p(x \mid c(x)=2)\cdot p(y \mid c(x)=2)\cdot p(c(x)=2)$$

$$= \begin{cases} p(x \mid c(x)=1)\cdot 0.8\cdot 0.8 + p(x \mid c(x)=2)\cdot 0.7\cdot 0.2 & if\ y=0 \\ p(x \mid c(x)=1)\cdot 0.2\cdot 0.8 + p(x \mid c(x)=2)\cdot 0.3\cdot 0.2 & if\ y=1 \end{cases}$$

$$= \begin{cases} 0.64\cdot X \sim N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}9 & 0\\0 & 1\end{bmatrix}\right) + 0.14\cdot X \sim N\left(\begin{bmatrix}2.5\\0\end{bmatrix}, \begin{bmatrix}9 & 0\\0 & 1\end{bmatrix}\right) & if\ y=0 \\ 0.16\cdot X \sim N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}9 & 0\\0 & 1\end{bmatrix}\right) + 0.06\cdot X \sim N\left(\begin{bmatrix}2.5\\0\end{bmatrix}, \begin{bmatrix}9 & 0\\0 & 1\end{bmatrix}\right) & if\ y=1 \end{cases}$$

$$p(x,y) = \begin{cases} X \sim N\left(\begin{bmatrix}0.35\\0\end{bmatrix}, \begin{bmatrix}3.8628 & 0\\0 & 0.4292\end{bmatrix}\right) & if\ y=0 \\ X \sim N\left(\begin{bmatrix}0.15\\0\end{bmatrix}, \begin{bmatrix}8.29\times10^{-4} & 0\\0 & 9.21\times10^{-5}\end{bmatrix}\right) & if\ y=1 \end{cases}$$

Question 2

1. Show that the solution for optimum weight $\tilde{w}$ still takes the similar form.

Starting from the loss function,

$$Loss = \frac{1}{2N}\sum_{i=1}^{n}(y_i - \phi(x_i)\cdot w)^2$$
$$Loss = \frac{1}{2N}\,(Y - \Phi\cdot w)^T\,(Y - \Phi\cdot w) \quad (converting\,to\,matrix)$$
$$\frac{dLoss}{dw} = \frac{1}{2N}\,(-2\Phi^T)\,(Y - \Phi\cdot w) \quad (differentiation)$$
$$0 = \Phi^T\cdot Y - \Phi^T\cdot\Phi\cdot w \quad (minimizing)$$
$$\Phi^T\cdot\Phi\cdot w = \Phi^T\cdot Y$$
$$w = (\Phi^T\cdot\Phi)^{-1}(\Phi^T\cdot Y)$$

2. Write $f(z)$ in terms of K and Y, where $k(x,z) = \phi(x)\cdot\phi(z)$

using $w = \Phi^T\cdot v$,

$$Loss = \frac{1}{2N}\,(Y - \Phi\cdot\Phi^T\cdot v)^T\,(Y - \Phi\cdot\Phi^T\cdot v)$$
$$\frac{dLoss}{dv} = \frac{1}{2N}\,(-2(\Phi\cdot\Phi^T)^T)\,(Y - \Phi\cdot\Phi^T\cdot v)$$
$$0 = (\Phi\cdot\Phi^T)^T\,(Y - \Phi\cdot\Phi^T\cdot v)$$

since $(\Phi\cdot\Phi^T)^T = (\Phi^T)^T\cdot(\Phi)^T = \Phi\cdot\Phi^T$,

$$0 = \Phi\cdot\Phi^T\,(Y - \Phi\cdot\Phi^T\cdot v)$$
$$0 = \Phi\cdot\Phi^T\cdot Y - \Phi\cdot\Phi^T\cdot\Phi\cdot\Phi^T\cdot v$$
$$\Phi\cdot\Phi^T\cdot\Phi\cdot\Phi^T\cdot v = \Phi\cdot\Phi^T\cdot Y$$
$$v = (\Phi\cdot\Phi^T\cdot\Phi\cdot\Phi^T)^{-1}\Phi^T\cdot\Phi\cdot Y$$

where

$$\Phi\cdot\Phi^T = \begin{bmatrix}\phi(x_1)\\ \phi(x_2)\\ \vdots\\ \phi(x_n)\end{bmatrix}\begin{bmatrix}\phi(x_1) & \phi(x_2) & \dots & \phi(x_n)\end{bmatrix}$$
$$= \begin{bmatrix}\phi(x_1)\cdot\phi(x_1) & \phi(x_1)\cdot\phi(x_2) & \dots & \phi(x_1)\cdot\phi(x_n)\\ \phi(x_2)\cdot\phi(x_1) & \phi(x_2)\cdot\phi(x_2) & \dots & \phi(x_2)\cdot\phi(x_n)\\ \vdots & \vdots & \ddots & \vdots\\ \phi(x_n)\cdot\phi(x_1) & \phi(x_n)\cdot\phi(x_2) & \dots & \phi(x_n)\cdot\phi(x_n)\end{bmatrix}$$
$$= \begin{bmatrix}\kappa(x_1,x_1) & \kappa(x_1,x_2) & \dots & \kappa(x_1,x_n)\\ \kappa(x_2,x_1) & \kappa(x_2,x_2) & \dots & \kappa(x_2,x_n)\\ \vdots & \vdots & \ddots & \vdots\\ \kappa(x_n,x_1) & \kappa(x_n,x_2) & \dots & \kappa(x_n,x_n)\end{bmatrix} = K$$
$$\Phi\cdot\Phi^T = K$$

therefore,

$$v = (K\cdot K)^{-1}\,(K\cdot Y)$$

$$f(z) = \phi(z)\cdot\Phi^T\cdot v$$

$$f(z) = \begin{bmatrix}\phi(z)\cdot\phi(x_1) & \phi(z)\cdot\phi(x_2) & \dots & \phi(z)\cdot\phi(x_n)\end{bmatrix}\cdot v$$

$$f(z) = \begin{bmatrix}\kappa(z,x_1) & \kappa(z,x_2) & \dots & \kappa(z,x_n)\end{bmatrix}\cdot v$$

$$f(z) = \begin{bmatrix}\kappa(z,x_1) & \kappa(z,x_2) & \dots & \kappa(z,x_n)\end{bmatrix}\cdot(K\cdot K)^{-1}\,(K\cdot Y)$$

Question 3

I. Linear Features

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

a) Write a routine that loads a data file and returns a matrix X containing all xi as rows, and a vector y containing all yi

In [2]:
```
def load_dataset(filename):
    x = []
    y = []
    with open(filename, 'r') as inputfile:
        for iline in inputfile:
            data = iline.strip('\n').split()
            # filling y
            y.append(float(data[-1]))
            # filling x
            x.append([float(xi) for xi in data[:-1]])
    X = np.asarray(x)
    Y = np.asarray(y)

    return X, Y

# testing
x1d, y1d = load_dataset('dataLinReg1D.txt')
x2d, y2d = load_dataset('dataLinReg2D.txt')
# print(x1d)
```

b) Write a routine that takes the raw X as input and returns a new X with a '1' appended to each row. This routine simply computes the linear features including the constant '1'. This routine can later be replaced by others to work with non-linear features.

Note: this notebook will use appended 1 (instead of pre-pend 1)

In [3]:
```
def linear_transform(X, D):
    return np.append(X, np.ones((X.shape[0],1)), axis=1)

# testing
x1d_appended = linear_transform(x1d, x1d)
# print(x1d_appended)
```

c) Write a routine that returns the optimal w from X and y - analytically, not by gradient descent.

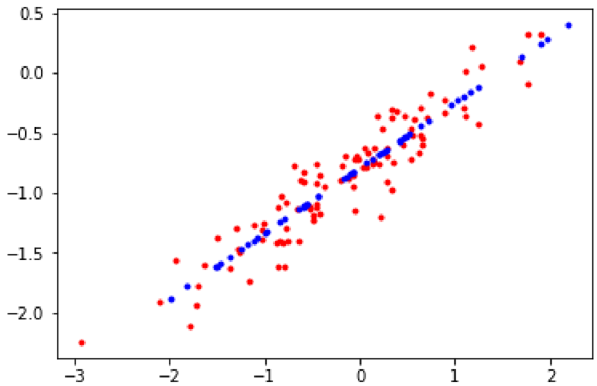
In [4]:
```
def ridge_regression(X, Y, lamb=1e-3):
    to_inv = np.matmul(X.T, X) + lamb * np.eye(X.shape[1])
#     print("to_inv:", to_inv.shape, np.linalg.det(to_inv))
    pre = np.linalg.inv(to_inv)
    post = np.matmul(X.T, Y)
    return np.matmul(pre, post)

# testing
w = ridge_regression(x1d_appended, y1d)
print(w)
```

```
[ 0.54359668 -0.79145962]
```

d) Generate some test data points (along a grid) and collect them in a matrix Z. Apply routine b) to compute features. Compute the predictions y = Zw (simple matrix multiplication) on the test data and plot it.

In [5]:
```
def plot_z_matrix(X, Y, count=50):
    Xi = linear_transform(X, X)
    w = ridge_regression(Xi, Y)
    Z = np.random.randn(count, 1)
    Zi = linear_transform(Z, Z)
    Yz = np.matmul(Zi, w)

    plt.plot(X, Y, '.r')
    plt.plot(Z, Yz, '.b')

# testing on 1d
print("Note: Blue points are from the samples Z, Red points are from the dataset")
plot_z_matrix(x1d, y1d)
```

```
Note: Blue points are from the samples Z, Red points are from the dataset
```



II. Cross-validation

In [6]:
```
x2d, y2d = load_dataset('dataLinReg2D.txt')

def rbf(x, y, alp=1e-3):
    norm = np.linalg.norm((x-y))
    return np.exp(-(norm/alp)**2)


def rbf_transform(X, D, alp=1e-3):
    rbfX = np.zeros((X.shape[0], D.shape[0]+1))

    for i in range(X.shape[0]):
        for j in range(D.shape[0]):
            rbfX[i, j] = rbf(X[i,:], D[j,:], alp=alp)
        rbfX[i, D.shape[0]] = 1

    return rbfX

# testing
rx1d = rbf_transform(x1d, x1d)
```

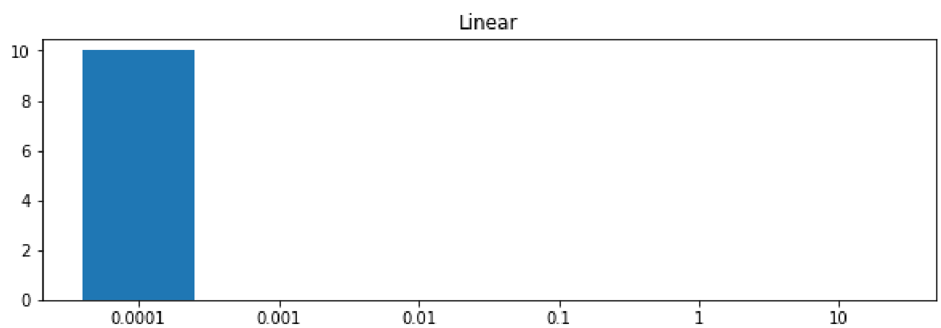
In [7]:
```
def loss(X, Y, w, lamb):
#     Yi = np.matmul(X, w)
    Yi = np.matmul(X,w)
    lse = 0.5 * np.linalg.norm(Yi - Y)**2
    ridge = lamb * np.linalg.norm(w)**2
    return lse + ridge


def cross_validate(X, Y, transform, shuffled_indices, folds=5, lamb=1e-3):
    N = X.shape[0]
    pn = int(np.ceil(N / folds))

    idx_pieces = [shuffled_indices[i*pn:min((i+1)*pn,N)] for i in range(folds)]

    Ws = []
    crossval_losses = []

    for i in range(folds):
#         print("crossval",i)
        idx_p = idx_pieces[:]
        del idx_p[i]

        Xt = X[np.concatenate(idx_p)]
        Xtrain = transform(Xt, Xt)
        Ytrain = Y[np.concatenate(idx_p)]

        Xv = X[idx_pieces[i]]
        Xval = transform(Xv, Xt)
        Yval = Y[idx_pieces[i]]

        w = ridge_regression(Xtrain, Ytrain, lamb=lamb)
#         print("w",w)
        Ws.append(w)

        cross_loss = loss(Xval, Yval, w, lamb)
        crossval_losses.append(cross_loss)

    return np.mean(crossval_losses)

# testing

x1d, y1d = load_dataset('dataLinReg1D.txt')

# randomize, then split
shuffled_indices = np.arange(x1d.shape[0])
np.random.shuffle(shuffled_indices)

CVLoss = cross_validate(x2d, y2d, linear_transform, shuffled_indices)
print('Linear:',CVLoss)
# ERROR - handle rbf pls
CVLoss = cross_validate(x2d, y2d, rbf_transform, shuffled_indices)
print('RBF:',CVLoss)
```

```
Linear: 0.11468528441796835
RBF: 17.336489677609876
```

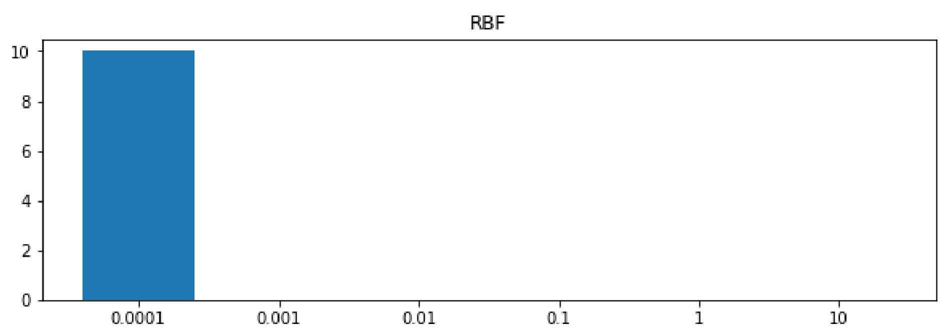
In [8]:
```
# lamb_list = [1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]

lamb_list = [1e-4, 1e-3, 1e-2, 1e-1, 1, 10]

opt_lamb = []
opt_cval = []

ropt_lamb = []
ropt_cval = []

for k in range(10):
    cval_list = []
    rcval_list = []

    shuffled_indices = np.arange(x2d.shape[0])
    np.random.shuffle(shuffled_indices)

    for l in lamb_list:
#         print(">> lambda",l)
        cval_list.append(cross_validate(x2d, y2d, linear_transform, shuffled_indices, lamb=l))
        rcval_list.append(cross_validate(x2d, y2d, rbf_transform, shuffled_indices, lamb=l))

    opt_idx = cval_list.index(min(cval_list))
    opt_lamb.append(lamb_list[opt_idx])
    opt_cval.append(min(cval_list))

    ropt_idx = rcval_list.index(min(rcval_list))
    ropt_lamb.append(lamb_list[ropt_idx])
    ropt_cval.append(min(rcval_list))
```

In [9]:
```
def bin_opt_lamb(opt, llist=lamb_list):
    opt_lamb_bin = []
    for l in llist:
        bin_count = 0
        for i in opt:
            if i == l:
                bin_count += 1
        opt_lamb_bin.append(bin_count)
    return opt_lamb_bin

op_lbin = bin_opt_lamb(opt_lamb)
rop_lbin = bin_opt_lamb(ropt_lamb)

def plot_bin(lbin, title, llist=lamb_list):
#     print(lbin)
    plt.figure(figsize=(10, 3))
    plt.bar([i for i in range(len(llist))], lbin, align='center', alpha=1)
    plt.title(title)
    plt.xticks([i for i in range(len(llist))], llist)
    plt.show()

print("Linear")
print('optimal lambda:',opt_lamb)
# print('cval losses:', opt_cval)
plot_bin(op_lbin, "Linear")

print("RBF")
print('optimal lambda:',ropt_lamb)
# print('cval losses:', ropt_cval)
plot_bin(rop_lbin, "RBF")
```

```
Linear
optimal lambda: [0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001]
```




```
RBF
optimal lambda: [0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001]
```



In [10]:
```
optimal_lambda = lamb_list[op_lbin.index(max(op_lbin))]
cvals = []
for i in range(10):
    shuffled_indices = np.arange(x2d.shape[0])
    np.random.shuffle(shuffled_indices)
    cvals.append(cross_validate(x2d, y2d, linear_transform, shuffled_indices, lamb=optimal_lambda))

opt_mean = np.mean(cvals)
opt_std = np.std(cvals)
print("Linear:\n  Optimal Lambda:",optimal_lambda,"\n  MSE mean:",opt_mean,"\n  MSE stdv:", opt_std)

roptimal_lambda = lamb_list[rop_lbin.index(max(rop_lbin))]
rcvals = []
for i in range(10):
    shuffled_indices = np.arange(x2d.shape[0])
    np.random.shuffle(shuffled_indices)
    rcvals.append(cross_validate(x2d, y2d, rbf_transform, shuffled_indices, lamb=roptimal_lambda))
opt_mean = np.mean(rcvals)
opt_std = np.std(rcvals)
print("\nRBF:\n  Optimal Lambda:",roptimal_lambda,"\n  MSE mean:",opt_mean,"\n  MSE stdv:", opt_std)
```

```
Linear:
  Optimal Lambda: 0.0001
  MSE mean: 0.10938421693716593
  MSE stdv: 0.0017477982036840799

RBF:
  Optimal Lambda: 0.0001
  MSE mean: 16.68747424286159
  MSE stdv: 0.15326666906242165
```

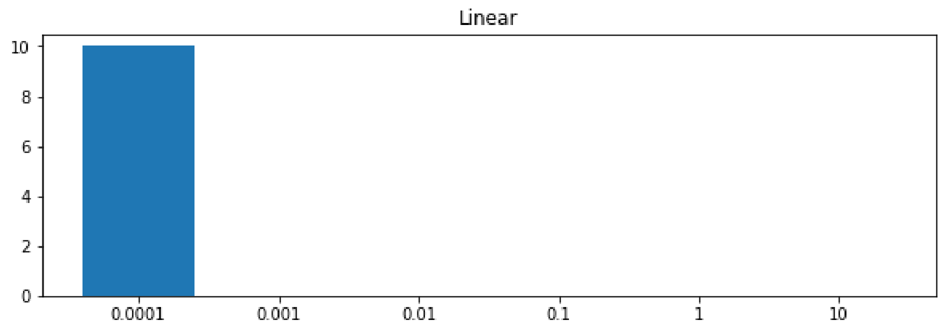
Adding noise $\sim N(0, 100)$

In [11]:
```
x2d, y2d_ = load_dataset('dataLinReg2D.txt')
ye = np.random.normal(0, 10, y2d.shape[0])
y2d = y2d_ + ye
```

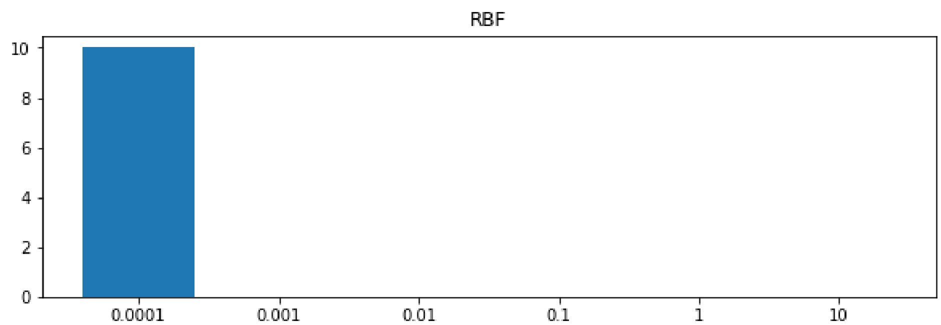
In [12]:
```
lamb_list = [1e-4, 1e-3, 1e-2, 1e-1, 1, 10]

opt_lamb = []
opt_cval = []

ropt_lamb = []
ropt_cval = []

for k in range(10):
    cval_list = []
    rcval_list = []

    shuffled_indices = np.arange(x2d.shape[0])
    np.random.shuffle(shuffled_indices)

    for l in lamb_list:
#        print(">> lambda",l)
        cval_list.append(cross_validate(x2d, y2d, linear_transform, shuffled_indices, lamb=l))
        rcval_list.append(cross_validate(x2d, y2d, rbf_transform, shuffled_indices, lamb=l))

    opt_idx = cval_list.index(min(cval_list))
    opt_lamb.append(lamb_list[opt_idx])
    opt_cval.append(min(cval_list))

    ropt_idx = rcval_list.index(min(rcval_list))
    ropt_lamb.append(lamb_list[ropt_idx])
    ropt_cval.append(min(rcval_list))
```

In [13]:
```
op_lbin = bin_opt_lamb(opt_lamb)
rop_lbin = bin_opt_lamb(ropt_lamb)

print("Linear")
print('optimal lambda:',opt_lamb)
# print('cval losses:', opt_cval)
plot_bin(op_lbin, "Linear")

print("RBF")
print('optimal lambda:',ropt_lamb)
# print('cval losses:', ropt_cval)
plot_bin(rop_lbin, "RBF")
```

```
Linear
optimal lambda: [0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001]
```




```
RBF
optimal lambda: [0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001]
```



In [14]:
```
optimal_lambda = lamb_list[op_lbin.index(max(op_lbin))]
cvals = []
for i in range(10):
    shuffled_indices = np.arange(x2d.shape[0])
    np.random.shuffle(shuffled_indices)
    cvals.append(cross_validate(x2d, y2d, linear_transform, shuffled_indices, lamb=optimal_lambda))

opt_mean = np.mean(cvals)
opt_std = np.std(cvals)
print("Linear:\n  Optimal Lambda:",optimal_lambda,"\n  MSE mean:",opt_mean,"\n  MSE stdv:", opt_std)

roptimal_lambda = lamb_list[rop_lbin.index(max(rop_lbin))]
rcvals = []
for i in range(10):
    shuffled_indices = np.arange(x2d.shape[0])
    np.random.shuffle(shuffled_indices)
    rcvals.append(cross_validate(x2d, y2d, rbf_transform, shuffled_indices, lamb=roptimal_lambda))
opt_mean = np.mean(rcvals)
opt_std = np.std(rcvals)
print("\nRBF:\n  Optimal Lambda:",roptimal_lambda,"\n  MSE mean:",opt_mean,"\n  MSE stdv:", opt_std)
```

```
Linear:
  Optimal Lambda: 0.0001
  MSE mean: 1111.5944621570166
  MSE stdv: 25.546773689208546

RBF:
  Optimal Lambda: 0.0001
  MSE mean: 1105.444180170073
  MSE stdv: 12.082771949084563
```

As can be seen, while the histogram of the optimal lambda doesn't change with the noise, the distribution of the cross-validation error within the optimal lambda adjust to the new values.