Machine Learning 1 WS18/19

Submission for Exercise Sheet 10

Michael Hoppe 362514
Wai Tang Victor Chan 406094
Jonas Piotrowski 399334
Aki Saksala 399293
Sourabh Raj 397371
Farnoush Rezaei Jafari 398708

1a) Suppose $X = \begin{bmatrix} | & & | \\ x_1 & \cdots & x_n \\ | & & | \end{bmatrix} \in \mathbb{R}^{d \times n}$, $y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n$

Then $\mathcal{E}_{RR}(w) = \|w^T X - y^T\|_2^2 + \lambda \|w\|_2^2$

$$= (w^T X - y^T)(w^T X - y^T)^T + \lambda w^T w$$

$$= (w^T X - y^T)(X^T w - y) + \lambda w^T w$$

$$= w^T X X^T w - y^T X^T w - w^T X y + y^T y + \lambda w^T w$$

these are just scalars so $y^T X^T w = w^T X y$

$$= w^T (X X^T + \lambda I) w - 2 w^T X y + y^T y$$

Set $\frac{d\mathcal{E}_{RR}}{dw} = 2(X X^T + \lambda I) w - 2 X y = 0$

$$\Rightarrow \quad w = (X X^T + \lambda I)^{-1} X y$$

1b) Kernalized problem:

$$\min_w \sum_{i=1}^n (w^T \phi(x_i) - y_i)^2 + \lambda \|w\|_2^2 \qquad (**)$$

Suppose $\phi(X) = \begin{bmatrix} | & & | \\ \phi(x_1) & \cdots & \phi(x_n) \\ | & & | \end{bmatrix}$

Then solution to $(**)$ is $w = (\phi(X)\phi(X)^T + \lambda I)^{-1} \phi(X) y$

Suppose we are given new data $x \in \mathbb{R}^n$ to make a prediction $\hat{y}$

Then
$$
\begin{aligned}
\hat{y} &= w^T \phi(x) \\
&= \phi(x)^T w \\
&= \phi(x)^T \left(\phi(X)\phi(X)^T + \lambda I \right)^{-1} \phi(X) y \\
&= \phi(x)^T \left(\phi(X)\phi(X)^T + \lambda I \right)^{-1} \underbrace{\phi(X) \left(\phi(X)^T \phi(X) + \lambda I \right)} \left(\phi(X)^T \phi(X) + \lambda I \right)^{-1} y
\end{aligned}
$$

$$
\begin{aligned}
&= \phi(X)\phi(X)^T\phi(X) + \lambda \phi(X) \\
&= \left(\phi(X)\phi(X)^T + \lambda I \right) \phi(X)
\end{aligned}
$$

$$
\begin{aligned}
&= \phi(x)^T \left(\phi(X)\phi(X)^T + \lambda I \right)^{-1} \left(\phi(X)\phi(X)^T + \lambda I \right) \phi(X) \left(\phi(X)^T\phi(X) + \lambda I \right)^{-1} y \\
&= \phi(x)^T \phi(X) \left(\phi(X)^T \phi(X) + \lambda I \right)^{-1} y \\
&= k^* \left(K + \lambda I \right)^{-1} y
\end{aligned}
$$

where $k^* \in \mathbb{R}^n$ with $[k^*]_i := k_\phi(x, x_i)$

$K \in \mathbb{R}^{n \times n}$ with $[K]_{ij} := k_\phi(x_i, x_j)$

$k_\phi(x, x') := \phi(x)^T \phi(x')$

It is then possible to use kernel function k to make prediction based on old data, without specifying ϕ.

Ex - 10

Ex - 2

1. $$\min_{\varepsilon, \omega} \sum_{i=1}^{n} \varepsilon_i^2$$

$$\text{st}: \quad \varepsilon_j = \omega^T x_j - y_j \quad \text{for} \quad 1 \le j \le n$$

$$\|\omega\|_2^2 \le c$$

Lagrange theorem:-

$$\mathcal{L}(\varepsilon, \omega, \alpha, \beta) = \frac{1}{2} \sum_{j=1}^{n} \varepsilon_j^2 + \sum_{j=1}^{n} \alpha_j (\varepsilon_j - \omega^T x_j - y_j) + \frac{1}{2} \beta (\|\omega\|^2 - c) \rightarrow (0)$$

$$\boxed{\max_{\alpha, \beta} \min_{\varepsilon, \omega} \mathcal{L}(\varepsilon, \omega, \alpha, \beta)} \Rightarrow \boxed{\beta \ge 0}$$

$$\frac{\partial \mathcal{L}}{\partial \varepsilon} = \sum_{j=1}^{n} \varepsilon_j + \sum_{j=1}^{n} \alpha_j = 0$$

$$\Rightarrow \boxed{\alpha_j = -\varepsilon_j} \quad \text{—} (1)$$

$$\frac{\partial \mathcal{L}}{\partial \omega} = -\sum_{j=1}^{n} \alpha_j x_j + \beta \omega = 0$$

$$\Rightarrow \boxed{\omega = \frac{1}{\beta} \sum_{j=1}^{n} \alpha_j x_j} \rightarrow (2)$$

Scanned by CamScanner

Replacing eq. (1) and (2) in eq (0)

$$L(\alpha, \beta) = \frac{1}{2}\sum_{j=1}^{n} \alpha_j^2 - \sum_{j=1}^{n} \alpha_j^2$$

$$- \sum_{j=1}^{n} \alpha_j \left(\frac{\sum_{j=1}^{n} \alpha_j x_j}{\beta} \right)^T x_j + \sum_{j=1}^{n} \alpha_j y_j$$

$$+ \frac{1}{2} \beta \left(\frac{\sum_{j=1}^{n}\sum_{j=1}^{n} \alpha_j \alpha_j x_j^T x_j}{\beta^2} - c \right)$$

$$\Rightarrow \boxed{-\frac{1}{2}\sum_{j=1}^{n} \alpha_j^2 - \frac{1}{2\beta}\sum_{j,j=1}^{n} \alpha_j \alpha_j x_j^T x_j + \sum_{j=1}^{n} \alpha_j y_j - \frac{\beta c}{2}} \rightarrow (3)$$

after matrix conversion and substituting,

$\boxed{x_j^T x_j = K}$ → Kernel

$$\boxed{L(\alpha, \beta) = -\frac{1}{2}\alpha^T\alpha - \frac{\alpha^T K \alpha}{2\beta} + \alpha^T y - \frac{\beta c}{2}} \rightarrow ($$

~~max(α,~~

$$\max_{\alpha, \beta} = L(\alpha, \beta)$$

$$\Rightarrow \frac{\partial L}{\partial \alpha} = -\alpha - \frac{K\alpha}{\beta} + y = 0$$

$$\Rightarrow \boxed{\alpha = y\left(I + \frac{K}{\beta}\right)^{-1}} \rightarrow (5)$$

⇒ Replacing (5) in eq. (2) :

$$\boxed{\alpha = y_i \left(K + \frac{1}{\beta} I \right)^{-1} y}$$ → (6)

<u>Results</u> :- Lagrange dual is eq. (4) and solution consists

$$\boxed{\alpha = y \left(I + \frac{K}{\beta} \right)^{-1}}$$ and $$\boxed{\beta \geq 0}$$

2. There are two ways to calculate the solution of primal program from dual.

1. → dual of dual program is the solution of primal program. So, performing Lagrange on dual equ. (4) would give primal program.

2. solution of ~~program~~ primal program can also be calculated using the solution of dual program and known <u>duality gap</u>. Basically the ~~max/m~~ optimal solution of

primal solution (p') and optional solution of dual solution (d') is ~~s~~ separated by duality gap.

$$\boxed{p' - d' = \text{duality gap}}$$

This duality gap is considered to be $\underline{\underline{0}}$ in case of strong duality and so,

$$\boxed{p' = d'}$$

(3) In $\underline{Q.1}$ we calculated

$$\boxed{w = x_i (K + \beta I)^{-1} y} \rightarrow (1)$$

→ ~~The function $f(x) =$~~

→ ~~or~~ $\underline{\text{Ridge regression}}$:-

$$f(x) = x^T w$$
$$= x^T x (K + \beta I)^{-1} y \rightarrow \text{from (1)}$$
$$\boxed{f(x) = K(x, x)(K + \beta I)^{-1} y}$$

→ $\underline{\text{Kernel ridge regression}}$:-

If we consider data in feature space eq. (1) can be re-written as

$$\boxed{\alpha = \phi(x) (K + \beta I)^{-1} y} \quad -(2)$$

so,

$$f(x) = \phi(x)^T \alpha$$

$$\boxed{f(x) = \phi(x)^T \phi(x) (K + \beta I)^{-1} y}$$

Scanned by CamScanner

Programming Part

January 13, 2019

1 Gaussian Processes

In this exercise, you will implement Gaussian process regression and apply it to a toy and a real dataset. We use the notation used in the paper "Rasmussen (2005). Gaussian Processes in Machine Learning" linked on ISIS.

Let us first draw a training set $X = (x_1, \ldots, x_n)$ and a test set $X_\star = (x_1^\star, \ldots, x_m^\star)$ from a d-dimensional input distribution. The Gaussian Process is a model under which the real-valued outputs $\mathbf{f} = (f_1, \ldots, f_n)$ and $\mathbf{f}_\star = (f_1^\star, \ldots, f_m^\star)$ associated to X and $X_\star$ follow the Gaussian distribution:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_\star \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_\star \\ \Sigma_\star^\top & \Sigma_{\star\star} \end{bmatrix} \right)$$

where

$$\begin{aligned} \Sigma &= k(X, X) + \sigma^2 I \\ \Sigma_\star &= k(X, X_\star) \\ \Sigma_{\star\star} &= k(X_\star, X_\star) + \sigma^2 I \end{aligned}$$

and where $k(\cdot, \cdot)$ is the Gaussian kernel function. (The kernel function is implemented in `utils.py`.) Predicting the output for new data points $X_\star$ is achieved by conditioning the joint probability distribution on the training set. Such conditional distribution called posterior distribution can be written as:

$$\mathbf{f}_\star | \mathbf{f} \sim \mathcal{N}(\underbrace{\Sigma_\star^\top \Sigma^{-1} \mathbf{f}}_{\mu_\star}\ ,\ \underbrace{\Sigma_{\star\star} - \Sigma_\star^\top \Sigma^{-1} \Sigma_\star}_{C_\star}) \tag{1}$$

Having inferred the posterior distribution, the log-likelihood of observing for the inputs $X_\star$ the outputs $\mathbf{y}_\star$ is given by evaluating the distribution $\mathbf{f}_\star | \mathbf{f}$ at $\mathbf{y}_\star$:

$$\log p(\mathbf{y}_\star | \mathbf{f}) = -\frac{1}{2} (\mathbf{y}_\star - \boldsymbol{\mu}_\star)^\top C_\star^{-1} (\mathbf{y}_\star - \boldsymbol{\mu}_\star) - \frac{1}{2} \log |C_\star| - \frac{m}{2} \log 2\pi \tag{2}$$

where $|\cdot|$ is the determinant. Note that the likelihood of the data given this posterior distribution can be measured both for the training data and the test data.

1.1 Part 1: Implementing a Gaussian Process (30 P)

Tasks:

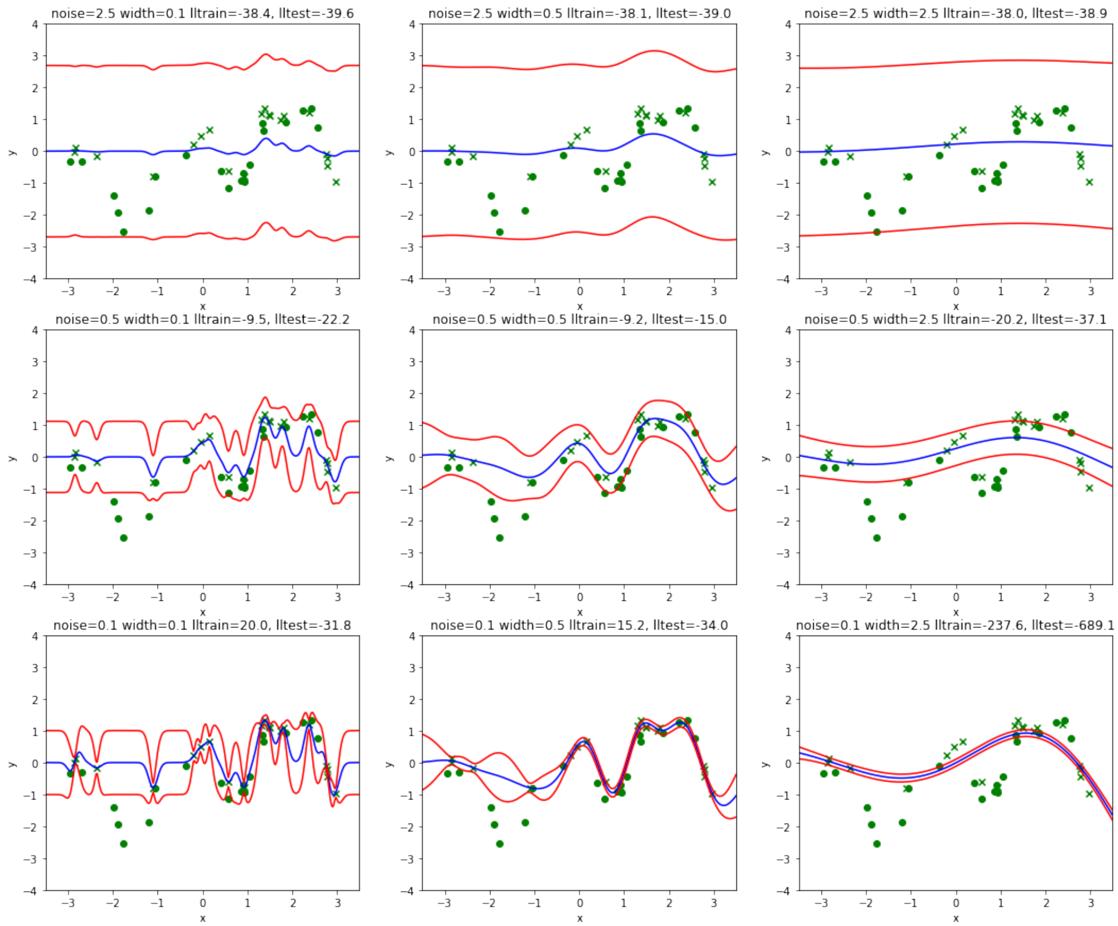
- **Create a class `GP_Regressor` that implements a Gaussian process regressor and has the following three methods:**
- **`def __init__(self,Xtrain,Ytrain,width,noise):`** Initialize a Gaussian process with noise parameter σ and width parameter w. The function must also precompute the matrix Σ^{-1} for subsequent use by the method `predict()` and `loglikelihood()`.
- **`def predict(self,Xtest):`** For the test set $X_\star$ of m points received as parameter, return the mean vector of size m and covariance matrix of size $m \times m$ of the corresponding output, that is, return the parameters $(\boldsymbol{\mu}_\star, C_\star)$ of the Gaussian distribution $\mathbf{f}_\star|\mathbf{f}$.
- **`def loglikelihood(self,Xtest,Ytest):`** For a data set $X_\star$ of m test points received as first parameter, return the loglikelihood of observing the outputs $\mathbf{y}_\star$ received as second parameter.

In [9]:
```
# --------------------------
# TODO: Replace by your code
# --------------------------
import utils
import numpy as np

class GP_Regressor():
    def __init__(self,Xtrain,Ytrain,width,noise):

        self.Xtrain = Xtrain
        self.Ytrain = Ytrain
        self.width = width
        self.noise = noise
        self.K = utils.gaussianKernel(Xtrain, Xtrain, width) + noise**2 * np.eye(len(X
        self.K_inv = np.linalg.inv(self.K)

    def predict(self,Xtest):

        K_s = utils.gaussianKernel(self.Xtrain, Xtest, width)
        K_ss = utils.gaussianKernel(Xtest, Xtest, width) + noise**2 * np.eye(len(Xtest

        self.mu_s = K_s.T.dot(self.K_inv).dot(self.Ytrain)
        self.cov_s = K_ss - K_s.T.dot(self.K_inv).dot(K_s)

        return self.mu_s, self.cov_s

    def loglikelihood(self,Xtest,Ytest):
        mean,cov = gp.predict(Xtest)
        tmp1, detcov = np.linalg.slogdet(self.cov_s)

        return -0.5 *((Ytest - self.mu_s).T.dot(np.linalg.inv(self.cov_s).dot((Ytest -
```

```
# --------------------------
```

- **Test your implementation by running the code below (it visualizes the mean and variance of the prediction at every location of the input space) and compares the behavior of the Gaussian process for various noise parameters σ and width parameters w.**

In [10]:
```
import utils,datasets,numpy
import matplotlib.pyplot as plt
%matplotlib inline

# Open the toy data
Xtrain,Ytrain,Xtest,Ytest = utils.split(*datasets.toy())

# Create an analysis distribution
Xrange = numpy.arange(-3.5,3.51,0.025)[:,numpy.newaxis]

f = plt.figure(figsize=(18,15))

# Loop over several parameters:
for i,noise in enumerate([2.5,0.5,0.1]):
    for j,width in enumerate([0.1,0.5,2.5]):

        # Create Gaussian process regressor object
        gp = GP_Regressor(Xtrain,Ytrain,width,noise)

        # Compute the predicted mean and variance for test data
        mean,cov = gp.predict(Xrange)
        var = cov.diagonal()

        # Compute the log-likelihood of training and test data
        lltrain = gp.loglikelihood(Xtrain,Ytrain)
        lltest  = gp.loglikelihood(Xtest ,Ytest )

        # Plot the data
        p = f.add_subplot(3,3,3*i+j+1)
        p.set_title('noise=%.1f width=%.1f lltrain=%.1f, lltest=%.1f'%(noise,width,ll
        p.set_xlabel('x')
        p.set_ylabel('y')
        p.scatter(Xtrain,Ytrain,color='green',marker='x') # training data
        p.scatter(Xtest,Ytest,color='green',marker='o')   # test data
        p.plot(Xrange,mean,color='blue')                  # GP mean
        p.plot(Xrange,mean+var**.5,color='red')           # GP mean + std
        p.plot(Xrange,mean-var**.5,color='red')           # GP mean - std
        p.set_xlim(-3.5,3.5)
        p.set_ylim(-4,4)
```



1.2 Part 2: Application to the Yacht Hydrodynamics Data Set (10 P)

In the second part, we would like to apply the Gaussian process regressor that you have implemented to a real dataset: the Yacht Hydrodynamics Data Set available on the UCI repository at the webpage http://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics. As stated on the web page, the input variables for this regression problem are:

1. Longitudinal position of the center of buoyancy
2. Prismatic coefficient
3. Length-displacement ratio
4. Beam-draught ratio
5. Length-beam ratio
6. Froude number

and we would like to predict from these variables the residuary resistance per unit weight of displacement (last column in the file `yacht_hydrodynamics.data`).

Tasks:

- **Load the data using `datasets.yacht()` and partition the data between training and test set using the function `utils.split()`. Normalize the data (center and rescale) so that the training data and labels have mean 0 and standard deviation 1 over the dataset for each dimension.**
- **Train several Gaussian processes on the regression task using various width and noise parameters.**
- **Draw two contour plots where the training and test log-likelihood are plotted as a function of the noise and width parameters. Choose suitable ranges of parameters so that the best parameter combination for the test set is in the plot. Use the same ranges and contour levels for training and test plots.**

In [150]:
```
# --------------------------
# TODO: Replace by your code
# --------------------------
%matplotlib inline
data = datasets.yacht()

Xtrain,Ytrain,Xtest,Ytest = utils.split(data[0],data[1])
Xtrain = Xtrain.reshape(153,6)
Xtest  = Xtest.reshape(154,6)

mean_x= Xtrain.mean(axis = 0)
std_x = Xtrain.std(axis=0)

Xtrain = (Xtrain - mean_x) / std_x
Xtest  = (Xtest - mean_x) / std_x


mean_y = Ytrain.mean(axis = 0)
std_y  = Ytrain.std(axis=0)

Ytrain =  (Ytrain - mean_y)   / std_y
Ytest =   (Ytest  - mean_y)   / std_y


print('After normalization:')
print ' '
print 'Xtrain.mean(axis = 0): %3.3f  %3.3f  %3.3f %3.3f  %3.3f %3.3f' %tuple(Xtrain.m
print 'Xtest.mean(axis = 0): %3.3f %3.3f %3.3f %3.3f %3.3f %3.3f' %tuple(Xtest.mean(a
print 'Xtrain.std(axis = 0):  %3.3f  %3.3f  %3.3f  %3.3f  %3.3f  %3.3f' %tuple(Xtrai
print 'Xtest.std(axis = 0):   %3.3f  %3.3f  %3.3f  %3.3f  %3.3f  %3.3f' %tuple(Xtest

print ' '

print 'Ytrain.mean(): %3.3f' %Ytrain.mean()
print 'Ytest.mean():   %3.3f' %Ytest.mean()
print 'Ytrain.std():   %3.3f' %Ytrain.std()
```

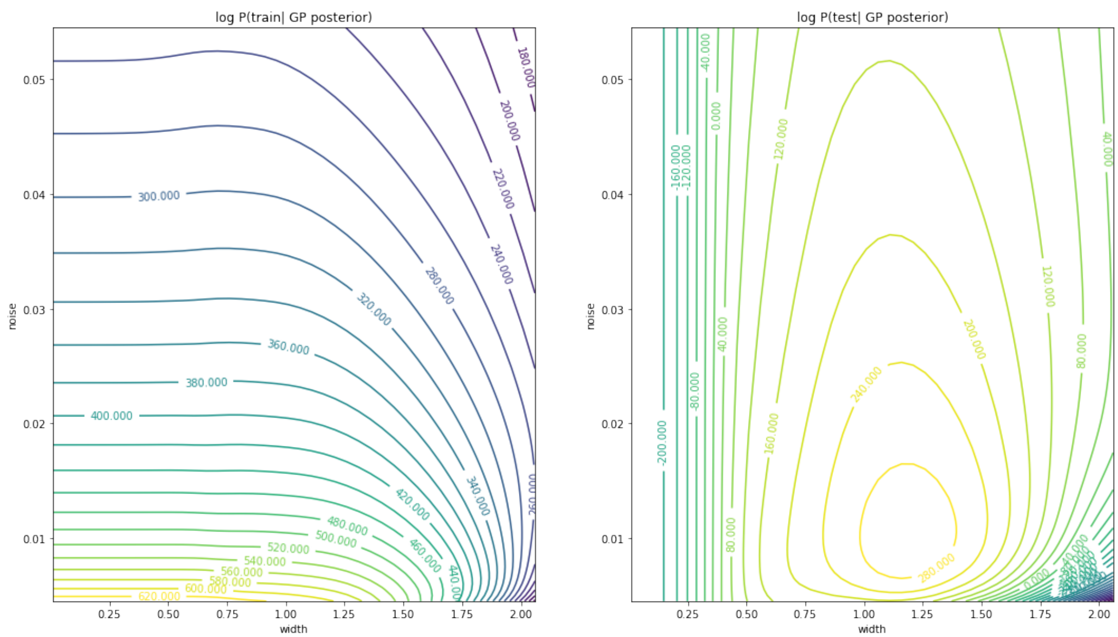
```
print 'Ytest.std():    %3.3f' %Ytest.std()



n = np.arange(0.0045,0.055,0.0005)
w = np.arange(0.01,2.1,0.05)

lltrain = np.zeros((n.size,w.size))
lltest  = np.zeros((n.size,w.size))

for i,noise in enumerate(n):
    for j,width in enumerate(w):



        # Create Gaussian process regressor object
        gp = GP_Regressor(Xtrain,Ytrain,width,noise)

        # Compute the predicted mean and variance for test data
        mean,cov = gp.predict(Xtest)
        #var = cov.diagonal()

        # Compute the log-likelihood of training and test data

        lltrain[i,j] = gp.loglikelihood(Xtrain,Ytrain)
        lltest[i,j]  = gp.loglikelihood(Xtest ,Ytest )



# --------------------------
```

```
After normalization:

Xtrain.mean(axis = 0): 0.000  0.000  0.000 -0.000  0.000 -0.000
Xtest.mean(axis = 0): -0.075 -0.177 -0.059 -0.031 -0.091 -0.001
Xtrain.std(axis = 0):  1.000  1.000  1.000  1.000  1.000  1.000
Xtest.std(axis = 0):   1.096  1.086  0.897  0.939  0.950  1.024

Ytrain.mean(): -0.000
Ytest.mean():   0.070
Ytrain.std():   1.000
Ytest.std():    1.091
```

In [156]:
```
plt.figure(figsize=(18,10))

plt.subplot(121)
c = plt.contour(w,n,lltrain,30)
```

```
        plt.title('log P(train| GP posterior)')
        plt.xlabel('width')
        plt.ylabel('noise')
        plt.clabel(c ,inline=1, fontsize=10)


        plt.subplot(122)
        d = plt.contour(w,n,lltest,30)
        plt.title('log P(test| GP posterior)')
        plt.xlabel('width')
        plt.ylabel('noise')
        plt.clabel(d ,inline=1, fontsize=10)
```

Out[156]: <a list of 25 text.Text objects>



In [ ]: