Regression_ML_professor

September 27, 2024

1 Parametric Model-Based regression

```
Notebook version: 1.4 (Sep 27, 2024)

Author: Jesús Cid-Sueiro (jesus.cid@uc3m.es)
        Jerónimo Arenas García (jarenas@tsc.uc3m.es)

Changes: v.1.0 - First version, expanding some cells from the Bayesian Regression
                 notebook
         v.1.1 - Python 3 version.
         v.1.2 - Revised presentation.
         v.1.3 - Updated index notation
         v.1.4 - Revised presentation order. Use of PolynomialFeatures. @ instead of dot. Defi

Pending changes: * Include regression on the stock data
```

[1]:
```
# Import some libraries that will be necessary for working with data and␣
 ↪displaying plots

# To visualize plots in the notebook
%matplotlib inline

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import scipy.io       # To read matlab files
import pylab

from sklearn.preprocessing import PolynomialFeatures
```

1.0.1 A quick note on the mathematical notation

In this notebook we will make extensive use of probability distributions. In general, we will use capital leters $\mathbf{X}$, S, E ..., to denote random variables, and lower-case letters $\mathbf{x}$, s, ϵ ..., to denote the values they can take.

In general, we will use letter p for probability density functions (pdf). When necessary, we will use, capital subindices to make the random variable explicit. For instance, $p_{\mathbf{X},S}(\mathbf{x}, s)$ would be the joint pdf of random variables $\mathbf{X}$ and S at values $\mathbf{x}$ and s, respectively.

However, to avoid a notation overload, we will omit subindices when they are clear from the context. For instance, we will use $p(\mathbf{x}, s)$ instead of $p_{\mathbf{X},S}(\mathbf{x}, s)$.

1.1 1. Model-based parametric regression

1.1.1 1.1. The regression problem

Given an observation vector $\mathbf{x}$, the goal of the regression problem is to find a function $f(\mathbf{x})$ providing *good* predictions about some unknown variable s. To do so, we assume that a set of *labelled* training examples, $\{\mathbf{x}_k, s_k\}_{k=0}^{K-1}$ is available.

The predictor function should make good predictions for new observations $\mathbf{x}$ not used during training. In practice, this is tested using a second set (the *test set*) of labelled samples.

1.1.2 1.2. The underlying model assumption

Many regression algorithms are grounded on the idea that all samples from the training set have been generated independently by some common stochastic process.

If $p(\mathbf{x}, s)$ were known, we could apply estimation theory to estimate s for a given $\mathbf{x}$ using p. For instance, we could apply any of the following classical estimates:

- Maximum A Posterior (MAP):

$$\hat{s}_{\text{MAP}} = \arg\max_s p(s|\mathbf{x})$$

- Minimum Mean Square Error (MSE):

$$\hat{s}_{\text{MSE}} = \mathbb{E}\{S|\mathbf{x}\} = \int s\, p(s|\mathbf{x})\, ds$$

Note that, since these estimators depend on $p(s|\mathbf{x})$, knowing the posterior distribution of the target variable is enough, and we do not need to know the joint distribution $p(\mathbf{x}, s)$.

More importantly, note that **if we knew the underlying model, we would not need the data** in D to make predictions on new data.

Exercise 1: Assume the target variable s is a scaled noisy version of the input variable x:

$$s = 2x + \epsilon$$

where ϵ is Gaussian a noise variable with zero mean and unit variance, which does not depend on x.

1. Compute the target model $p(s|x)$
2. Compute prediction $\hat{s}_{\text{MAP}}$ for an arbitrary input x
3. Compute prediction $\hat{s}_{\text{MSE}}$ for an arbitrary input x
4. Compute prediction $\hat{s}_{\text{MSE}}$ for input $x = 4$

Solution:

1. Since ϵ is Gaussian, so it is s for a given x, with mean

$$\mathbb{E}\{S \mid x\} = \mathbb{E}\{2x \mid x\} + \mathbb{E}\{\epsilon \mid x\} = 2x + 0 = 2x$$

   and variance

$$\mathrm{Var}\{S \mid x\} = \mathrm{Var}\{2x \mid x\} + \mathrm{Var}\{\epsilon \mid x\} = 0 + 1 = 1$$

   therefore

$$p(s|x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(s - 2x)^2\right)$$

2. The MAP estimate is

$$\hat{s}_{\mathrm{MAP}} = \arg\max_s p(s|x) = 2x$$

3. Since the MSE estimate is the conditional mean, which has been already computed, have $\hat{s}_{\mathrm{MSE}} = 2x$
4. The prediction is $\hat{s}_{\mathrm{MSE}} = 2 \cdot 4 = 8$

1.1.3 1.3. Model-based regression

In practice, the underlying model is usually unknown.

Model based-regression methods exploit the idea of using the training data to estimate the posterior distribution $p(s|\mathbf{x})$ and then apply estimation theory to make predictions.

1.1.4 1.4. Parametric model-based regression

In some cases, we may have a partial knowledge about the underlying mode. In this notebook we will assume that p belongs to a parametric family of distributions $p(s|\mathbf{x}, \mathbf{w})$, where $\mathbf{w}$ is some unknown parameter.

Exercise 2: Assume the target variable s is a scaled noisy version of the input variable x:

$$s = wx + \epsilon$$

where ϵ is Gaussian a noise variable with zero mean and unit variance, which does not depend on x. Assume that w is known. 1. Compute the target model $p(s|x, w)$ 2. Compute prediction $\hat{s}_{\mathrm{MAP}}$ for an arbitrary input x 3. Compute prediction $\hat{s}_{\mathrm{MSE}}$ for an arbitrary input x

Solution:

1. As in Exercise 1 ϵ is Gaussian, so it is s for a given x, with mean

$$\mathbb{E}\{S|x, w\} = wx$$

   and variance

$$\mathrm{Var}\{S|x, w\} = 1$$

   therefore

$$p(s|x, w) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(s - wx)^2\right)$$

2. The MAP estimate is
$$\hat{s}_{\text{MAP}} = \arg\max_s p(s|x, w) = wx$$
3. Since the MSE estimate is the conditional mean, which has been already computed, have $\hat{s}_{\text{MSE}} = wx$

We will use the training data to estimate **w**

The estimation of **w** from a given dataset $\mathcal{D}$ is the goal of the following sections

1.2 2. Maximum Likelihood parameter estimation.

The ML (Maximum Likelihood) principle is well-known in statistics and can be stated as follows: take the value of the parameter to be estimated (in our case, **w**) that best explains the given observations (in our case, the training dataset $\mathcal{D}$). Mathematically, this can be expressed as follows:

$$\hat{\mathbf{w}}_{\text{ML}} = \arg\max_{\mathbf{w}} p(\mathcal{D}|\mathbf{w})$$

Exercise 3: All samples in dataset $D = \{(x_k, s_k), k = 0, \dots, K-1\}$

$$s_k = w \cdot x_k + \epsilon_k$$

where ϵ_k are i.i.d. (independent and identically distributed) Gaussian noise random variables with zero mean and unit variance, which do not depend on x_k.

Compute the ML estimate, $\hat{w}_{\text{ML}}$, of w.

Solution: From Exercise 2,

$$p\left(s_k \mid x_k, w\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(s_k - wx_k\right)^2\right)$$

Since the noise variables are i.i.d, we have

$$p(\mathcal{D}|w) = p\left(s_0, \dots, s_{K-1} \mid x_0, \dots, x_{K-1}, w\right) p\left(x_0, \dots, x_{K-1} \mid w\right) \tag{1}$$
$$= \prod_{k=0}^{K-1} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(s_k - wx_k\right)^2\right) p\left(x_0, \dots, x_{K-1}\right) \tag{2}$$
$$= \frac{1}{(2\pi)^{\frac{K}{2}}} \exp\left(-\frac{1}{2}\sum_{k=0}^{K-1}\left(s_k - wx_k\right)^2\right) p\left(x_0, \dots, x_{K-1}\right) \tag{3}$$

Therefore:

$$\hat{\mathbf{w}}_{\text{ML}} = \arg\max_{\mathbf{w}} p(\mathcal{D}|\mathbf{w}) = \arg\min_{\mathbf{w}} \sum_{k=0}^{K-1}\left(s_k - wx_k\right)^2$$

Differentiating with respect to w:

$$-2\sum_{k=0}^{K-1}\left(s_k - \hat{w}_{\text{ML}} x_k\right) x_k = 0$$

that is

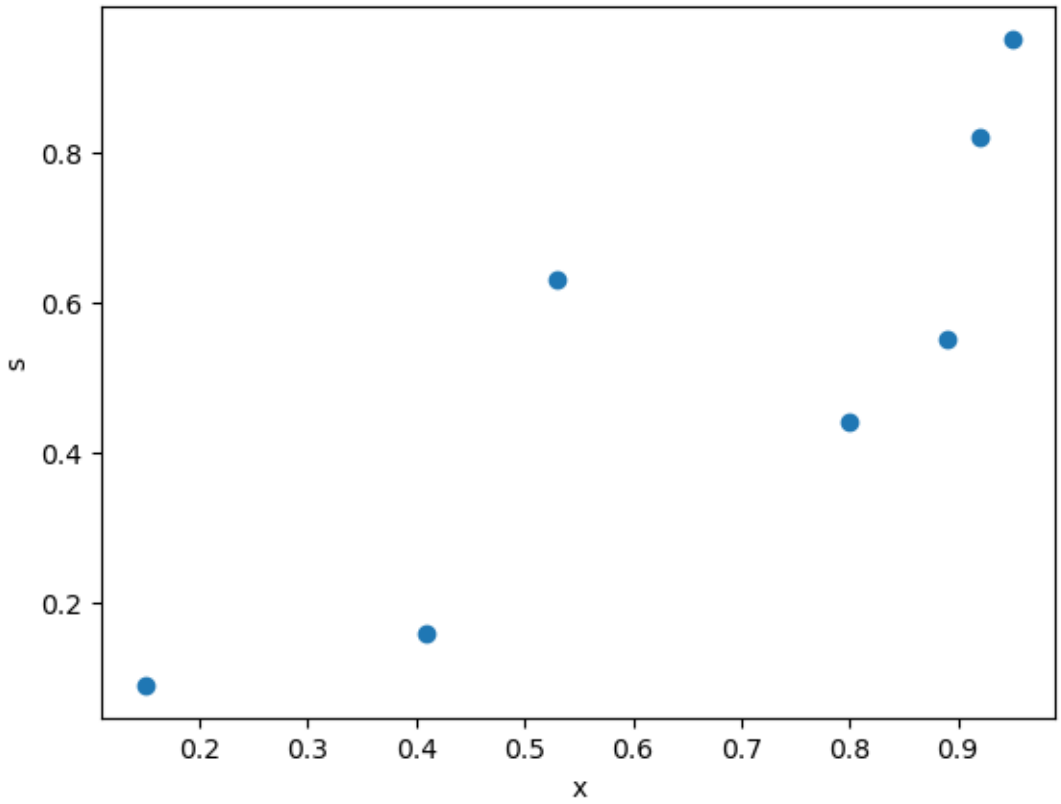
$$\hat{w}_{\text{ML}} = \frac{\sum_{k=0}^{K-1} s_k x_k}{\sum_{k=0}^{K-1}\left(x_k\right)^2}$$

Exercise 4: The inputs and the targets from a dataset D have been stored in the following Python arrays:

[2]:
```
X = np.array([0.15, 0.41, 0.53, 0.80, 0.89, 0.92, 0.95])
s = np.array([0.09, 0.16, 0.63, 0.44, 0.55, 0.82, 0.95])
```

- **4.1.** Represent a scatter plot of the data points

[3]:
```
# <SOL>
plt.figure()
plt.scatter(X, s)
plt.xlabel('x')
plt.ylabel('s')
plt.show()
# </SOL>
```



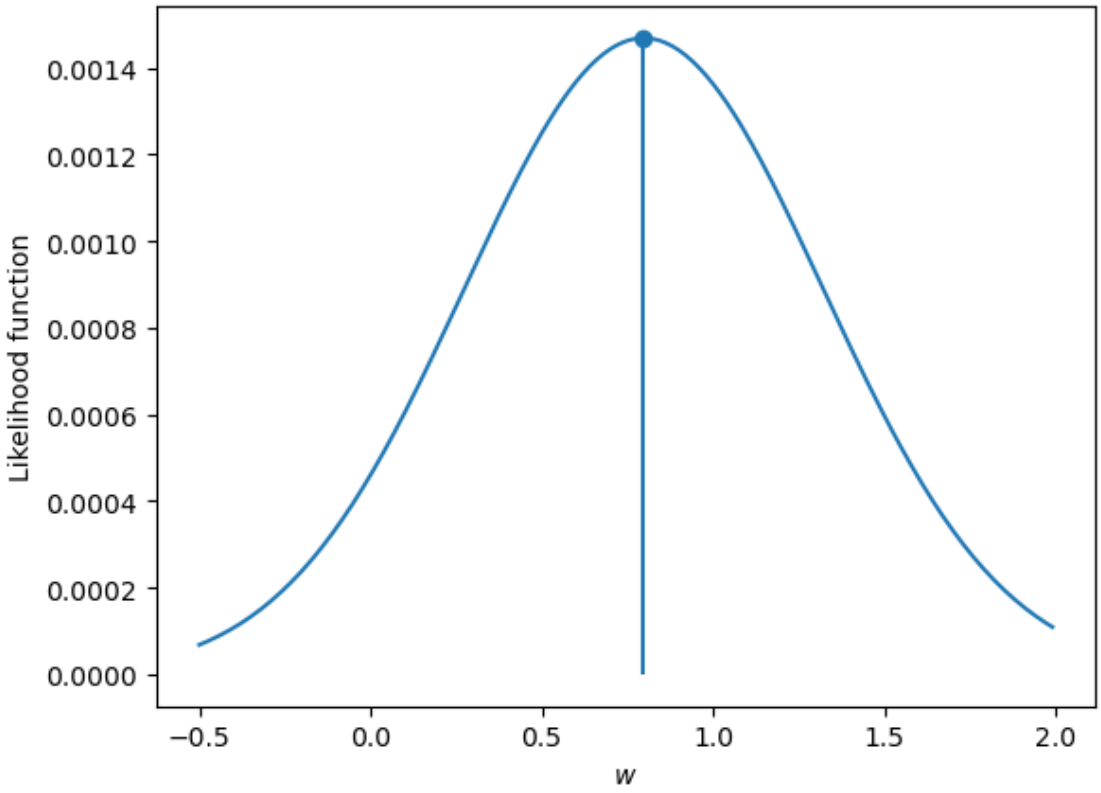
- **4.2.** Compute the ML estimate

[4]:
```
# wML = <FILL IN>
wML = np.sum(X*s) / np.sum(X*X)

print("The ML estimate is {}".format(wML))
```

```
The ML estimate is 0.79709787816564
```

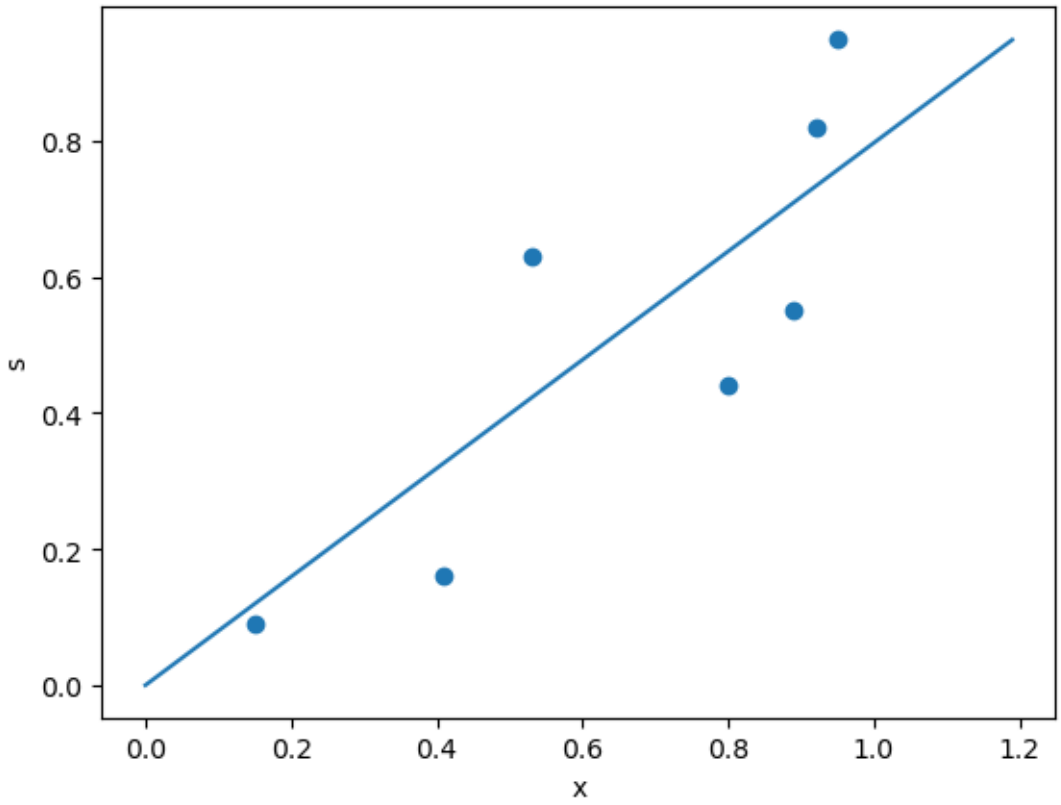
- **4.3.** Plot the likelihood as a function of parameter w along the interval $-0.5 \leq w \leq 2$, verifying that the ML estimate takes the maximum value.

[5]:
```
sigma_eps = 1
K = len(s)
wGrid = np.arange(-0.5, 2, 0.01)

p = []
for w in wGrid:
    d = s - X*w
    # p.append(<FILL IN>)
    p.append((1.0/(np.sqrt(2*np.pi)*sigma_eps))**K * np.exp(-np.dot(d, d) /␣
 ↪(2*sigma_eps**2)))

# Compute the likelihood for the ML parameter wML
# d = <FILL IN>
d = s-X*wML
# pML = [<FILL IN>]
pML = [(1.0/(np.sqrt(2*np.pi)*sigma_eps))**K * np.exp(-np.dot(d, d) /␣
 ↪(2*sigma_eps**2))]

# Plot the likelihood function and the optimal value
plt.figure()
plt.plot(wGrid, p)
plt.stem([wML], pML)
plt.xlabel('$w$')
plt.ylabel('Likelihood function')
plt.show()
```



- **4.4.** Plot the prediction function on top of the data scatter plot

[6]:
```
xgrid = np.arange(0, 1.2, 0.01)
# sML = <FILL IN>
sML = wML * xgrid

plt.figure()
plt.scatter(X, s)
# plt.plot(<FILL IN>)
plt.plot(xgrid, sML)
plt.xlabel('x')
plt.ylabel('s')
plt.axis('tight')
plt.show()
```



1.2.1 2.1. Model assumptions

In order to solve exercise 4 we have taken advantage of the statistical independence of the noise components. Some independence assumptions are required in general to compute the ML estimate in other scenarios.

In order to estimate $\mathbf{w}$ from the training data in a mathematicaly rigorous and compact form let us group the target variables into a vector

$$\mathbf{s} = (s_0, \dots, s_{K-1})^\top$$

and the input vectors into a matrix

$$\mathbf{X} = (\mathbf{x}_0, \dots, \mathbf{x}_{K-1})^\top$$

We will make the following assumptions:

- A1. All samples in D have been generated by the same distribution, $p(\mathbf{x}, s \mid \mathbf{w})$
- A2. Input variables $\mathbf{x}$ do not depend on $\mathbf{w}$. This implies that

$$p(\mathbf{X} \mid \mathbf{w}) = p(\mathbf{X})$$

- A3. Targets $s_0, \ldots, s_{K-1}$ are statistically independent, given $\mathbf{w}$ and the inputs $\mathbf{x}_0, \ldots, \mathbf{x}_{K-1}$, that is:

$$p(\mathbf{s} \mid \mathbf{X}, \mathbf{w}) = \prod_{k=0}^{K-1} p(s_k \mid \mathbf{x}_k, \mathbf{w})$$

Since $D = (\mathbf{X}, \mathbf{s})$, we can write

$$p(\mathcal{D}|\mathbf{w}) = p(\mathbf{s}, \mathbf{X}|\mathbf{w}) = p(\mathbf{s}|\mathbf{X}, \mathbf{w})p(\mathbf{X}|\mathbf{w})$$

Using assumption A2,

$$p(\mathcal{D}|\mathbf{w}) = p(\mathbf{s}|\mathbf{X}, \mathbf{w})p(\mathbf{X})$$

and, finally, using assumption A3, we can express the estimation problem as the computation of

$$\begin{aligned}
\hat{\mathbf{w}}_{\mathrm{ML}} &= \arg\max_{\mathbf{w}} p(\mathbf{s}|\mathbf{X}, \mathbf{w}) && (4)\\
&= \arg\max_{\mathbf{w}} \prod_{k=0}^{K-1} p(s_k \mid \mathbf{x}_k, \mathbf{w}) && (5)\\
&= \arg\max_{\mathbf{w}} \sum_{k=0}^{K-1} \log p(s_k \mid \mathbf{x}_k, \mathbf{w}) && (6)\\
&= \arg\max_{\mathbf{w}} \mathrm{L}(\mathbf{w}) && (7)
\end{aligned}$$

where

$$\mathrm{L}(\mathbf{w}) = \sum_{k=0}^{K-1} \log p(s_k \mid \mathbf{x}_k, \mathbf{w}) \quad (8)$$

is usually named the **log-likelihood** function. The logarithm transform products into sums, and exponentials (very common in distribution models) into their exponents, which usually simplifies the optimization.

The ML estimate is also frequently expressed as a function of the negative log-likelihood function, that is

$$\hat{\mathbf{w}}_{\mathrm{ML}} = \arg\min_{\mathbf{w}} \mathrm{NLL}(\mathbf{w}) \quad (9)$$

where $\mathrm{NLL}(\mathbf{w}) = -L(\mathbf{w})$.

1.2.2 2.2. Summary.

Let's summarize what we need to do in order to design a regression algorithm based on ML estimation:

1. Assume a parametric data model $p(s|\mathbf{x}, \mathbf{w})$
2. Using the data model and the i.i.d. assumption, compute $p(\mathbf{s}|\mathbf{X}, \mathbf{w})$.
3. Find an expression for $\mathbf{w}_{\mathrm{ML}}$
4. Assuming $\mathbf{w} = \mathbf{w}_{\mathrm{ML}}$, compute the MAP or the minimum MSE estimate of s given $\mathbf{x}$.

Exercise 5: Assume the dataset $\mathcal{D} = \{(x_k, s_k, k = 0, \dots, K-1\}$ contains i.i.d. samples from a distribution with posterior density given by

$$p(s \mid x, w) = wx \exp(-wxs), \qquad s \geq 0,\ x \geq 0,\ w \geq 0$$

- **5.1.** For an arbitrary w, compute the prediction function based on the estimate s_{MSE}

Solution:

$$\hat{s}_{\text{MSE}} = \mathbb{E}\{s|x, w\} = \int swx \exp(wxs) ds = \frac{1}{wx}$$

- **5.2.** Determine an expression for the negative log-likelihood function

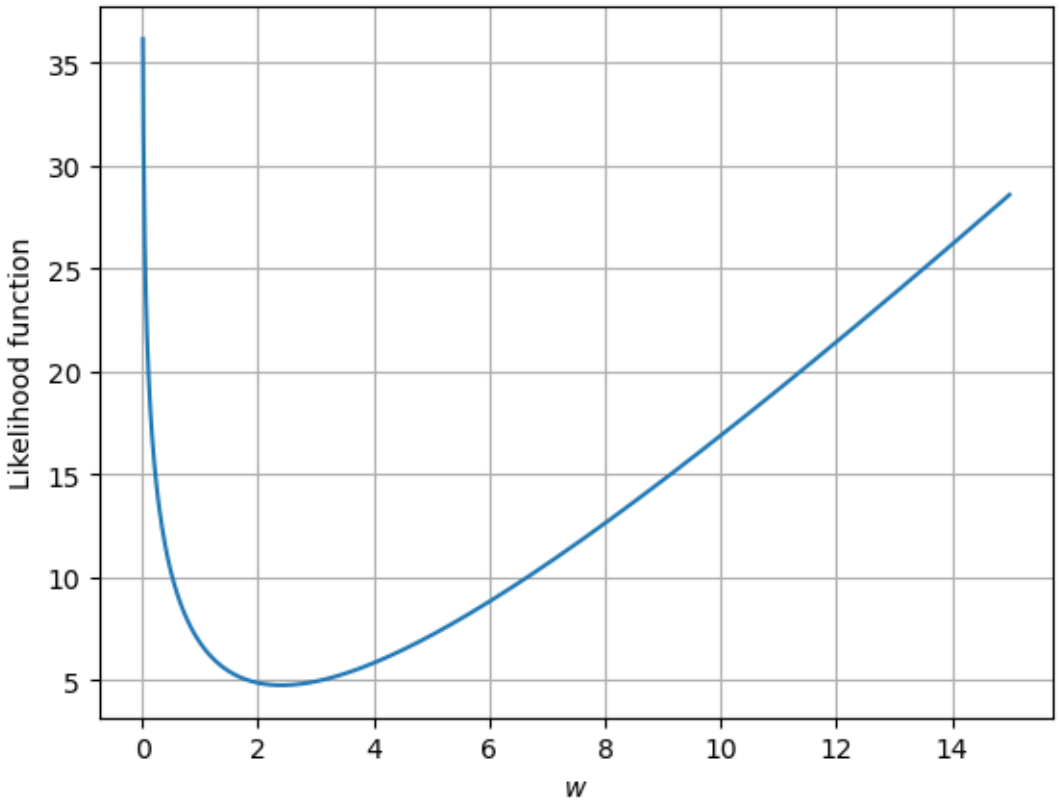
Solution: The negative log-likelihood function is

$$p(\mathbf{s}|w, \mathbf{X}) = \prod_{k=0}^{K-1} wx_k \exp(-wx_k s_k) = w^K \left(\prod_{k=0}^{K-1} x_k \right) \exp\left(-w \sum_{k=0}^{K-1} x_k s_k \right)$$

$$\text{NLL}(w) = -\sum_{k=0}^{K-1} \log\left(wx_k \exp(-wx_k s_k)\right) = -K \log(w) - \sum_{k=0}^{K-1} \log\left(x_k\right) + w \sum_{k=0}^{K-1} x_k s_k$$

- **5.3.** Draw the negative log-likelihood function for the dataset in **Exercise 4** in the range $0.01 \leq w \leq 15$.

[7]:
```
K = len(s)
wGrid = np.arange(0.01, 15, 0.01)

nll = []
Lx = np.sum(np.log(X))
xs = np.dot(X,s)
for w in wGrid:
    # nll.append(<FILL IN>)
    nll.append(- K * np.log(w) - Lx + w * xs)

plt.figure()
# plt.plot(<FILL IN>)
plt.plot(wGrid, nll)
plt.xlabel('$w$')
plt.ylabel('Likelihood function')
plt.grid()
plt.show()
```



- **5.4.** Determine the maximum likelihood parameter, w_{ML}.

(*Hint: you can maximize the log-likelihood function instead of the likelihood function in order to simplify the differentiation*)

Solution:

From the plot, we can see that the NLL is minimum at the unique stationary point. The derivative of the NLL is

$$\frac{d\mathrm{NLL}(w)}{dw} = \frac{K}{w} - \mathbf{X}^\top \mathbf{s}$$

which is zero for

$$w_{\mathrm{ML}} = \frac{K}{\mathbf{X}^\top \mathbf{s}}$$

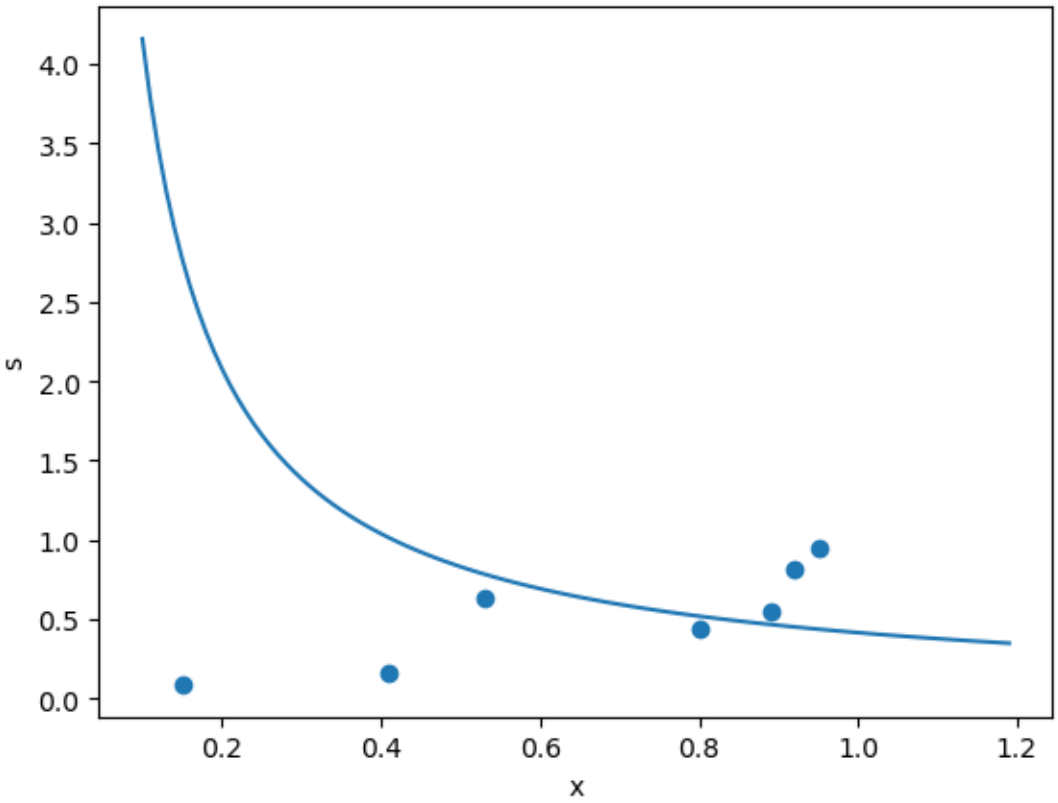
- **5.5.** Compute w_{ML} for the dataset in **Exercise 4**

[8]:
```
# wML = <FILL IN>
wML = float(K) /xs
print(wML)
```

```
2.4043415538915984
```

- **5.6.** Plot the prediction function obtained in ap. 5.1 for $w = w_{\mathrm{ML}}$, and compare it with the linear predictor in exercise 4

[9]:
```
xgrid = np.arange(0.1, 1.2, 0.01)
# sML = <FILL IN>
sML = 1 / (wML * xgrid)

plt.figure()
plt.scatter(X, s)
# plt.plot(<FILL IN>)
plt.plot(xgrid, sML)
plt.xlabel('x')
plt.ylabel('s')
plt.axis('tight')
plt.show()
```



Subjectively, we can see that the predictor computed in exercise 6 does not fit the given data very well. This could be a false perception. If the data have been truly generated by the parametric model assumed in exercise 6 (i.e. $p(s \mid x, w) = wx \exp(-wxs)$, the apparent missbehavior of the estimator could be caused by the natural randomness of the data, and a greater amount of data would show a better adjustement.

However, maybe the parametric model chosen for $p(s|x, w)$ is not appropriate: while the data show that s tends to grow with x, the prediction function computed in exercise 5.1 is decreasing for all values of parameter w it its domain.

On the contrary, the gaussian model in exercise 4 has, at least, captured the positive correlation between x and s.

In summary, the choice of the data model is important. In many applications, no parametric data model is available, and the data scientist must make a choice based on the nature of the data or any previous knowledge about the statistical behavior of the data.

If no previous information is available, the data scientist can try different models, and compare using validation data and some cross validation technique.

1.3 3. ML estimation for a Gaussian model.

The Gaussian model in exercise 4 can be generalized to generate non-linear and multi-dimensional regression functions.

1.3.1 3.1. Step 1: The Gaussian generative model

Let us assume that the target variables s_k in dataset $\mathcal{D}$ are given by

$$s_k = \mathbf{w}^\top \mathbf{z}_k + \varepsilon_k$$

where $\mathbf{z}_k$ is the result of some transformation of the inputs, $\mathbf{z}_k = T(\mathbf{x}_k)$, and ε_k are i.i.d. instances of a Gaussian random variable with mean zero and varianze σ_ε^2, i.e.,

$$p_E(\varepsilon) = \frac{1}{\sqrt{2\pi}\sigma_\varepsilon} \exp\left(-\frac{\varepsilon^2}{2\sigma_\varepsilon^2}\right)$$

Assuming that the noise variables are independent of $\mathbf{x}$ and $\mathbf{w}$, then, the target variable is conditionally gaussian with mean $\mathbf{w}^\top \mathbf{z}_k$ and variance σ_ε^2,

$$p(s|\mathbf{x}, \mathbf{w}) = p_E(s - \mathbf{w}^\top \mathbf{z}) = \frac{1}{\sqrt{2\pi}\sigma_\varepsilon} \exp\left(-\frac{(s - \mathbf{w}^\top \mathbf{z})^2}{2\sigma_\varepsilon^2}\right)$$

1.3.2 3.2. Step 2: Likelihood and negative log-likelihood functions

Now we need to compute the likelihood function $p(\mathbf{s}, \mathbf{X}|\mathbf{w})$. If the samples are i.i.d. we can write

$$p(\mathbf{s}|\mathbf{X}, \mathbf{w}) = \prod_{k=0}^{K-1} p(s_k|\mathbf{x}_k, \mathbf{w}) \tag{10}$$

$$= \prod_{k=0}^{K-1} \frac{1}{\sqrt{2\pi}\sigma_\varepsilon} \exp\left(-\frac{\left(s_k - \mathbf{w}^\top \mathbf{z}_k\right)^2}{2\sigma_\varepsilon^2}\right) \tag{11}$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma_\varepsilon}\right)^K \exp\left(-\frac{1}{2\sigma_\varepsilon^2} \sum_{k=1}^{K} \left(s_k - \mathbf{w}^\top \mathbf{z}_k\right)^2\right) \tag{12}$$

Finally, grouping variables $\mathbf{z}_k$ in

$$\mathbf{Z} = \left(\mathbf{z}_0, \dots, \mathbf{z}_{K-1}\right)^\top$$

we get

$$p(\mathbf{s}|\mathbf{X}, \mathbf{w}) = \left(\frac{1}{\sqrt{2\pi}\sigma_\varepsilon}\right)^K \exp\left(-\frac{1}{2\sigma_\varepsilon^2} \|\mathbf{s} - \mathbf{Z}\mathbf{w}\|^2\right)$$

Therefore

$$\text{NLL}(\mathbf{w}) = K \log\left(\sqrt{2\pi}\sigma_\varepsilon\right) + \frac{1}{2\sigma_\varepsilon^2}\|\mathbf{s} - \mathbf{Z}\mathbf{w}\|^2$$

1.3.3 3.3. Step 3: ML estimation.

The maximum likelihood solution is then given by:

$$\mathbf{w}_{ML} = \arg\min_{\mathbf{w}} \text{NLL}(\mathbf{w}) = \arg\min_{\mathbf{w}} \|\mathbf{s} - \mathbf{Z}\mathbf{w}\|^2$$

Note that $\|\mathbf{s} - \mathbf{Z}\mathbf{w}\|^2$ is the sum or the squared prediction errors (Sum of Squared Errors, SSE) for all samples in the dataset. For this reason, this is also called the ***Least Squares*** (LS) solution.

The LS solution can be easily computed by differentiation,

$$\nabla_{\mathbf{w}} \|\mathbf{s} - \mathbf{Z}\mathbf{w}\|^2 \Big|_{\mathbf{w}=\mathbf{w}_{\text{ML}}} = -2\mathbf{Z}^\top \mathbf{s} + 2\mathbf{Z}^\top \mathbf{Z}\mathbf{w}_{\text{ML}} = \mathbf{0}$$

and it is equal to

$$\mathbf{w}_{\text{ML}} = (\mathbf{Z}^\top \mathbf{Z})^{-1}\mathbf{Z}^\top \mathbf{s}$$

1.3.4 3.4. Step 4: Prediction function.

The last step consists on computing an estimate of s by assuming that the true value of the weight parameters is $\mathbf{w}_{\text{ML}}$. In particular, the minimum MSE estimate is

$$\hat{s}_{\text{MSE}} = \mathbb{E}\{s|\mathbf{x}, \mathbf{w}_{\text{ML}}\}$$

Knowing that, given $\mathbf{x}$ and $\mathbf{w}$, s is normally distributed with mean $\mathbf{w}^\top \mathbf{z}$ we can write

$$\hat{s}_{\text{MSE}} = \mathbf{w}_{\text{ML}}^\top \mathbf{z}$$

Exercise 6: Assume that the targets in the one-dimensional dataset in exercise 4 have been generated by the polynomial Gaussian model

$$s = w_0 + w_1 x + w_2 x^2 + \epsilon$$

(i.e., with $\mathbf{z} = T(x) = (1, x, x^2)^\top$) with noise standard deviation

[10]:
```
sigma_eps = 0.3
```

- **6.1.** Compute the ML estimate for the given datasets. To generate the data matriz $\mathbf{Z}$, you might like to use the `PolynomialFeatures` class from `sklearn`.

[11]:
```
# Compute the extended input matrix Z
# g = <FILL IN>
g = 2    # Polynomial degree
# Create the nonlinear transformation object from sklearn
poly = PolynomialFeatures(g)
```

```
# Apply the transformation to the input data using the fit_transform method
# Note that we need to reshape the input data X to a 2D array
# Z = <FILL IN>
Z = poly.fit_transform(X[:,np.newaxis])

# Compute the ML estimate using linalg.lstsq from Numpy.
# wML = <FILL IN>
wML = np.linalg.lstsq(Z, s, rcond=None)[0]
print(wML)
```

```
[0.01643973 0.50472594 0.32250412]
```

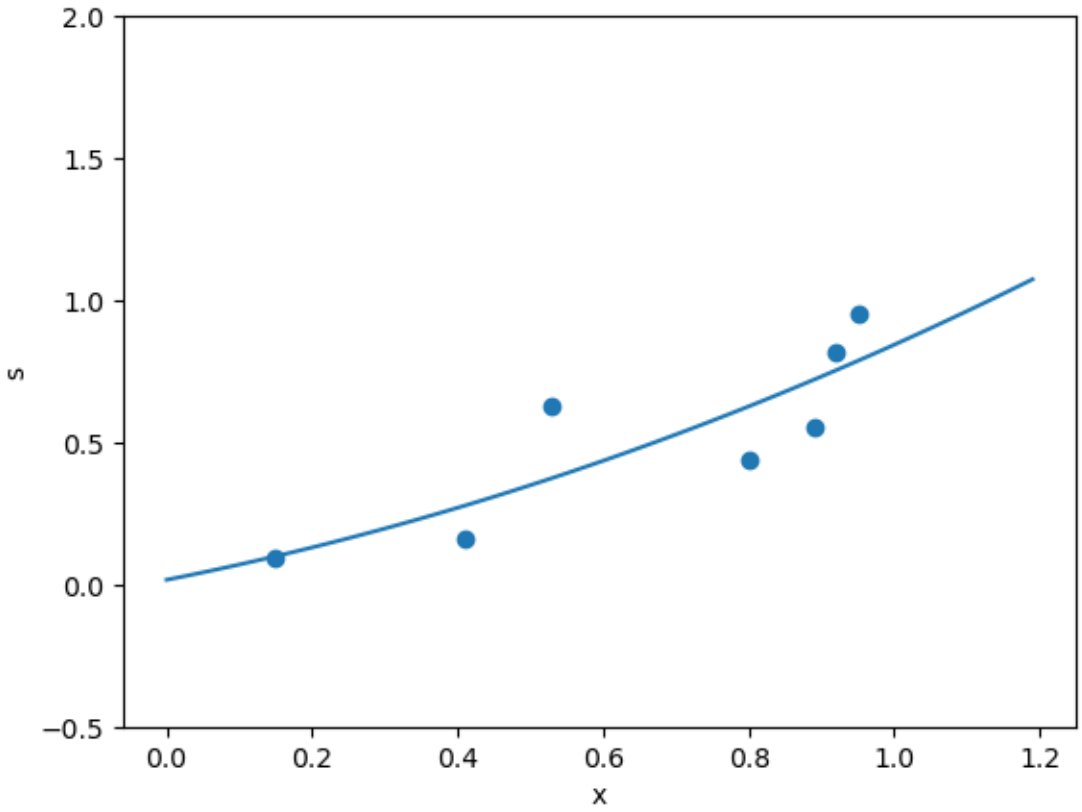
- **6.2.** Compute the value of the log-likelihood function for $\mathbf{w} = \mathbf{w}_{\text{ML}}$.

[12]:
```
K = len(s)

# Compute the likelihood for the ML parameter wML
# d = <FILL IN>
d = s - Z @ wML

# LwML = [<FILL IN>]
LwML = - K/2*np.log(2*np.pi*sigma_eps**2) - np.dot(d, d) / (2*sigma_eps**2)

print(LwML)
```

```
1.0272256462076963
```

- **6.3.** Plot the prediction function over the data scatter plot

[13]:
```
xgrid = np.arange(0, 1.2, 0.01)
nx = len(xgrid)
# Apply the polynomial transformation to the grid data in x
# Z = <FILL IN>
Z = poly.fit_transform(xgrid[:,np.newaxis])

# sML = <FILL IN>
sML = Z @ wML

plt.figure()
plt.scatter(X, s)
# plt.plot(<FILL IN>)
plt.plot(xgrid, sML)
plt.xlabel('x')
plt.ylabel('s')
plt.ylim([-0.5, 2])
plt.show()
```



You might like to explore the prediction function for higher polynomial degrees, to get a betted adjustment of the data. The higher the degree, the better the approximation, but the higher the risk of overfitting. Once again, in order to select the appropriate degree, we would need to apply a cross-validation procedure.