ENM 360: Introduction to Data-driven Modeling

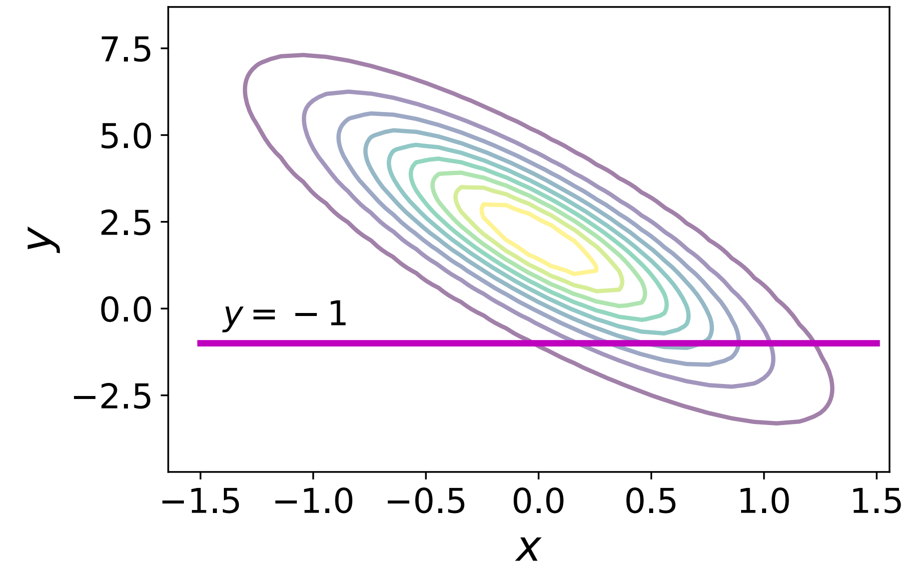
Lecture #8: Linear regression

Paris Perdikaris
September 24, 2020

Penn
UNIVERSITY *of* PENNSYLVANIA

Recap: Marginals and conditionals of a Gaussian

$$p(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy}\end{bmatrix}\right)$$



Conditional distribution

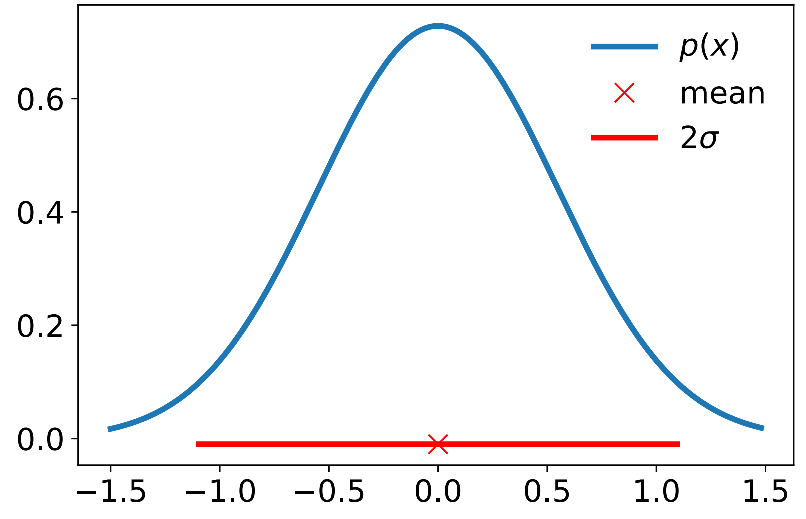
$$p(\boldsymbol{x} \mid \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{\mu}_{x \mid y}, \boldsymbol{\Sigma}_{x \mid y}\right)$$

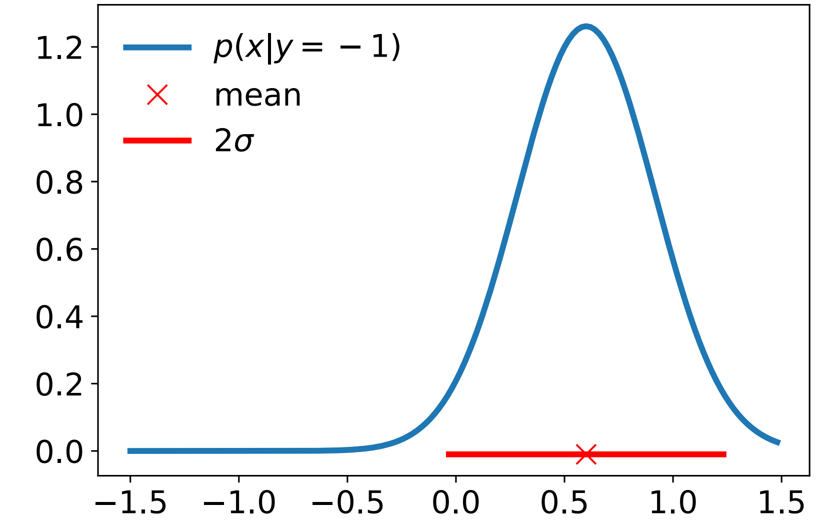
$$\boldsymbol{\mu}_{x \mid y} = \boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_y)$$

$$\boldsymbol{\Sigma}_{x \mid y} = \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\,.$$

Marginal distribution

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y} = \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx}\right)$$





These are unique properties that make the Gaussian distribution very simple and attractive to compute with! It is essentially our main building block for computing under uncertainty.

Kernel density estimation in SciPy

scipy.stats.gaussian_kde

class scipy.stats.**gaussian_kde**(*dataset*, *bw_method=None*) [source]

Representation of a kernel-density estimate using Gaussian kernels.

Kernel density estimation is a way to estimate the probability density function (PDF) of a random variable in a non-parametric way. gaussian_kde works for both uni-variate and multi-variate data. It includes automatic bandwidth determination. The estimation works best for a unimodal distribution; bimodal or multi-modal distributions tend to be oversmoothed.

Parameters:

dataset : *array_like*

Datapoints to estimate from. In case of univariate data this is a 1-D array, otherwise a 2-D array with shape (# of dims, # of data).

bw_method : *str, scalar or callable, optional*

The method used to calculate the estimator bandwidth. This can be 'scott', 'silverman', a scalar constant or a callable. If a scalar, this will be used directly as *kde.factor*. If a callable, it should take a gaussian_kde instance as only parameter and return a scalar. If None (default), 'scott' is used. See Notes for more details.

Kernel density estimation in SciPy

```
>>> from scipy import stats
>>> def measure(n):
...     "Measurement model, return two coupled measurements."
...     m1 = np.random.normal(size=n)
...     m2 = np.random.normal(scale=0.5, size=n)
...     return m1+m2, m1-m2
```

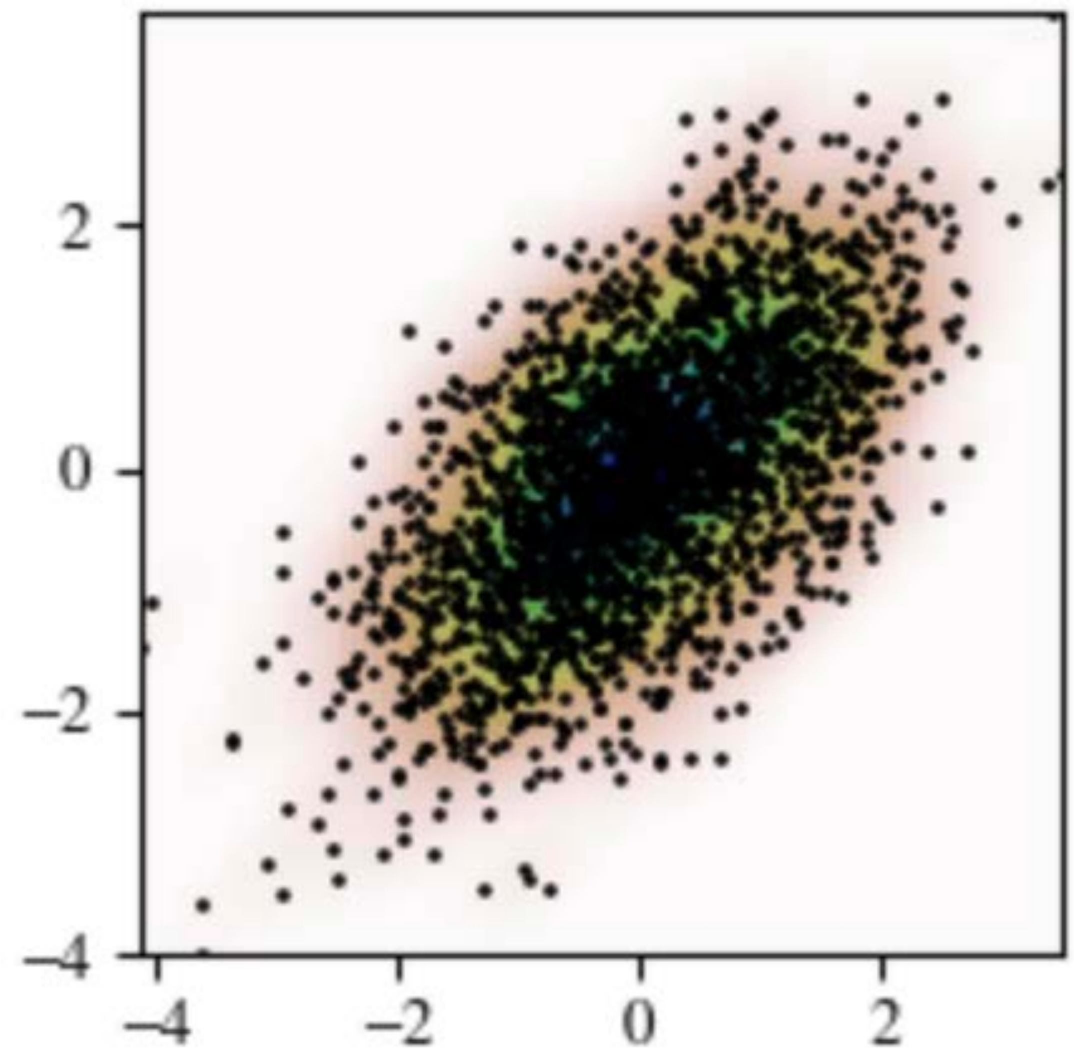
```
>>> m1, m2 = measure(2000)
>>> xmin = m1.min()
>>> xmax = m1.max()
>>> ymin = m2.min()
>>> ymax = m2.max()
```

Perform a kernel density estimate on the data:

```
>>> X, Y = np.mgrid[xmin:xmax:100j, ymin:ymax:100j]
>>> positions = np.vstack([X.ravel(), Y.ravel()])
>>> values = np.vstack([m1, m2])
>>> kernel = stats.gaussian_kde(values)
>>> Z = np.reshape(kernel(positions).T, X.shape)
```

Plot the results:

```
>>> import matplotlib.pyplot as plt
>>> fig, ax = plt.subplots()
>>> ax.imshow(np.rot90(Z), cmap=plt.cm.gist_earth_r,
...           extent=[xmin, xmax, ymin, ymax])
>>> ax.plot(m1, m2, 'k.', markersize=2)
>>> ax.set_xlim([xmin, xmax])
>>> ax.set_ylim([ymin, ymax])
>>> plt.show()
```



$$f : \mathcal{X} \to \mathcal{Y}$$

Supervised learning

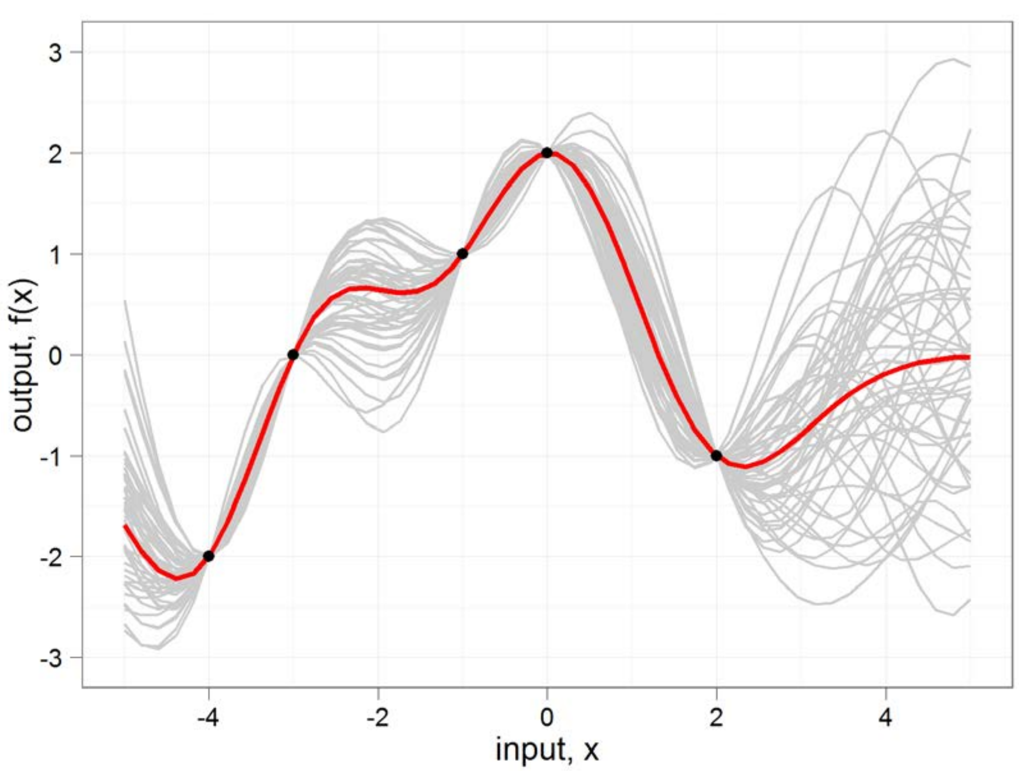
$$f : \mathcal{X} \to \mathcal{Y}$$

$$\mathcal{D} = \{\boldsymbol{x}, \boldsymbol{y}\},\ \boldsymbol{x} \in \mathcal{X},\ \ \boldsymbol{y} \in \mathcal{Y}$$

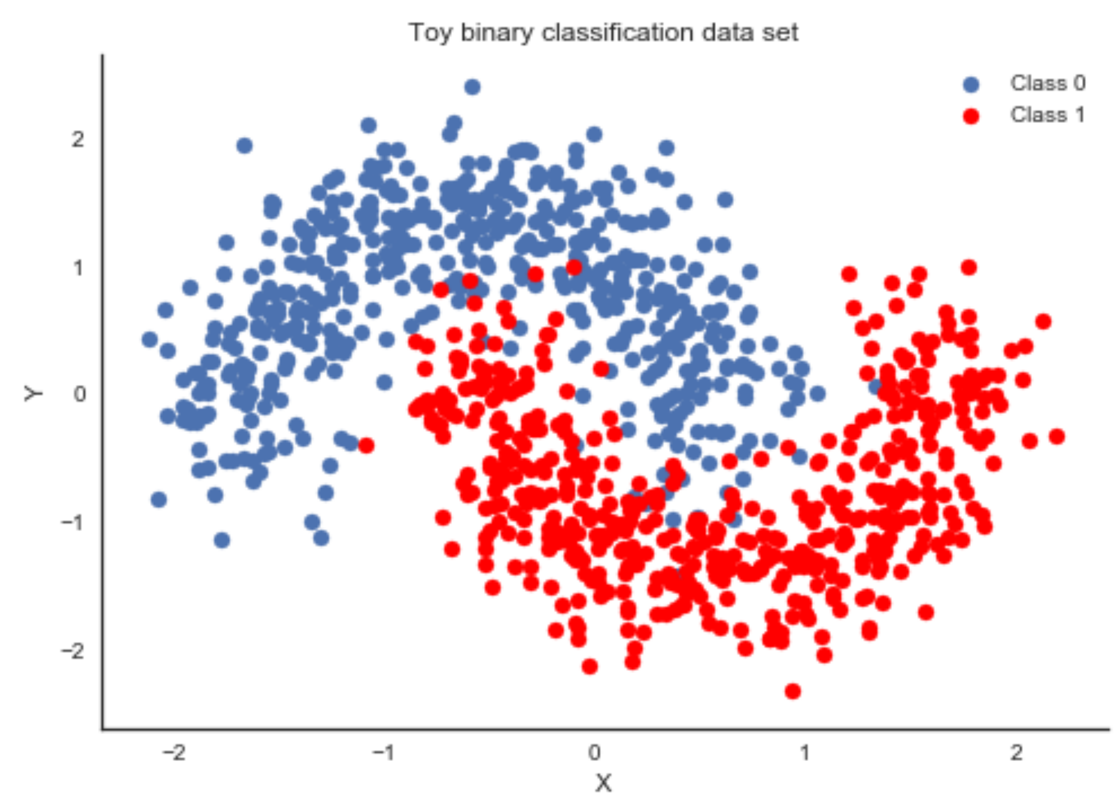
$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon$$

$$p(f(\boldsymbol{x}^*)|\boldsymbol{x}^*, \mathcal{D})$$

Regression



Classification



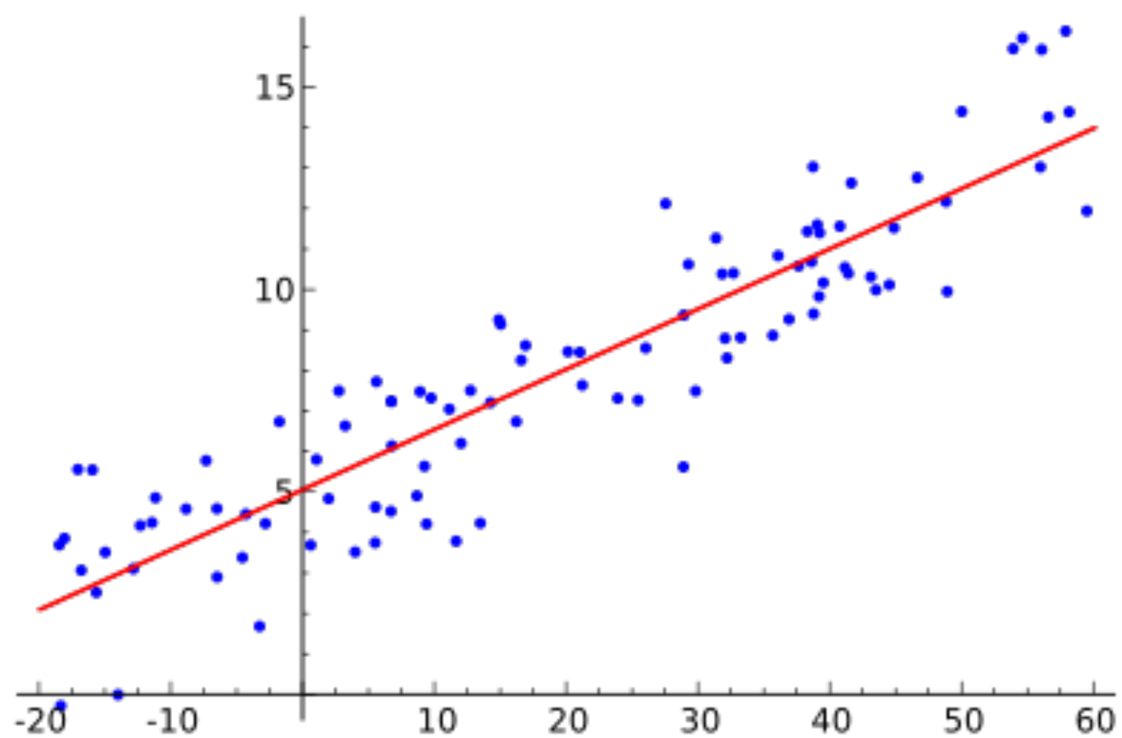
Linear regression

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

$$\mathcal{D} = \{\boldsymbol{x}, \boldsymbol{y}\},\ \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} \in \mathcal{Y}$$

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon$$

$$f(\boldsymbol{x}) = w^T \boldsymbol{x}$$



"It's <u>not</u> just about lines and planes!"

Linear regression with basis functions

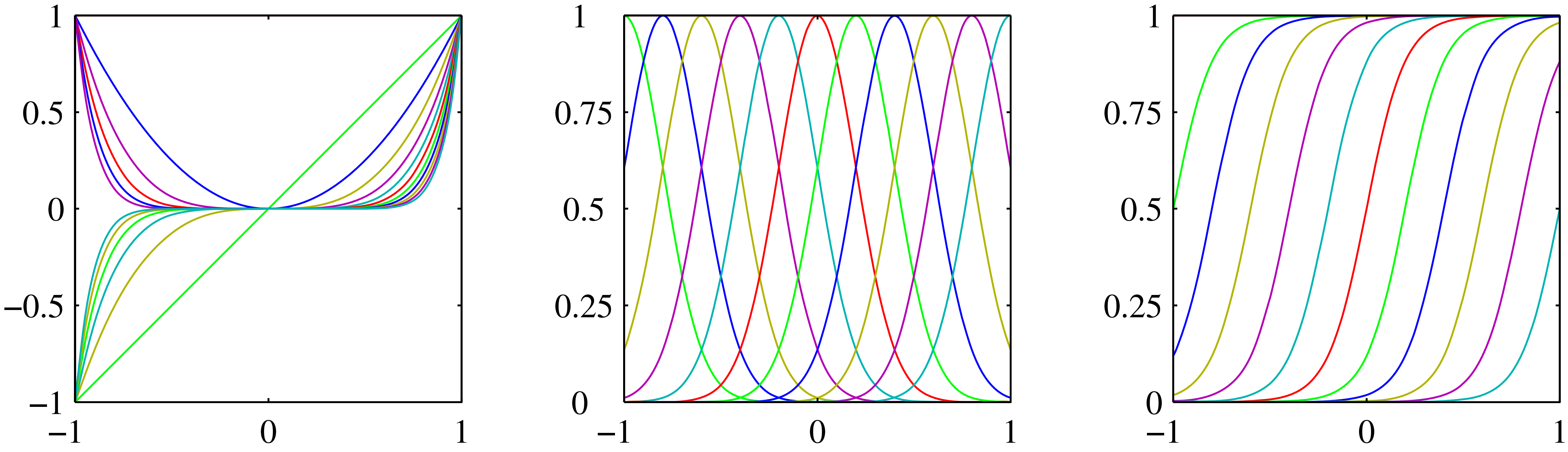
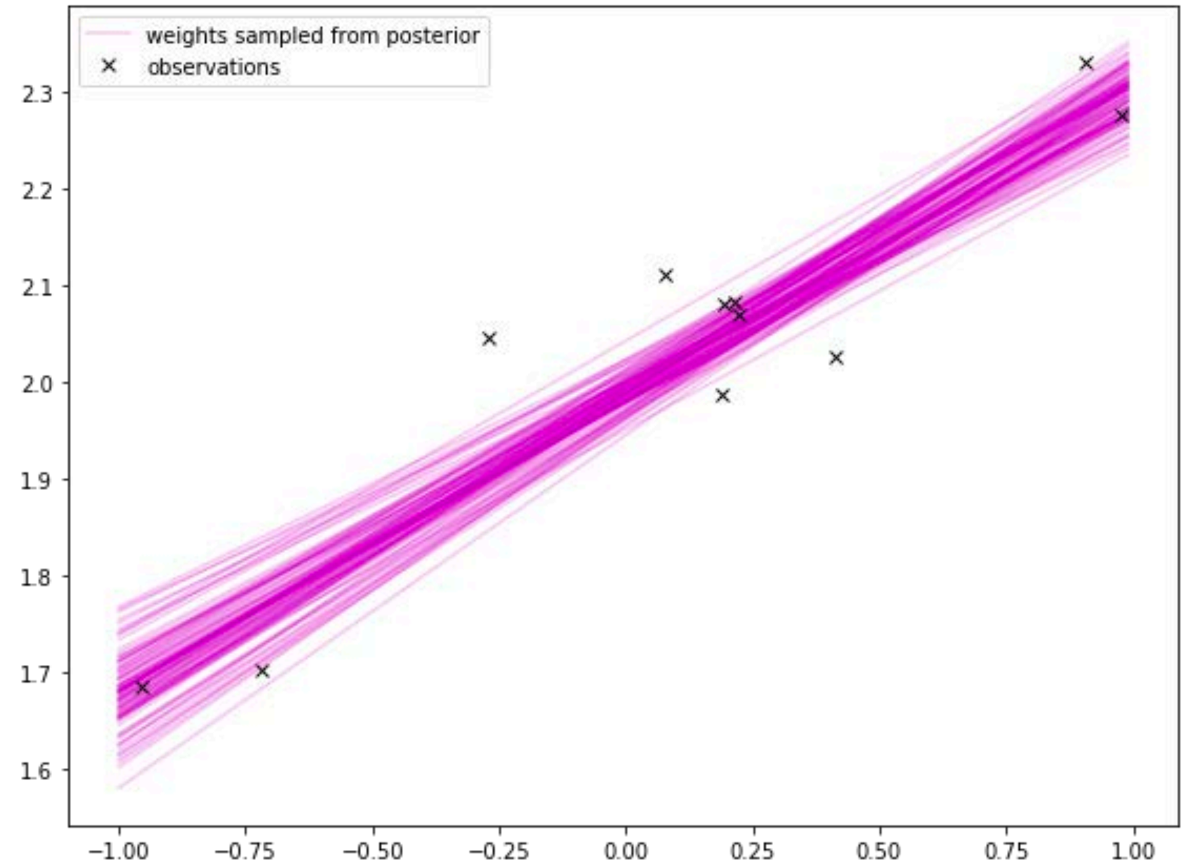
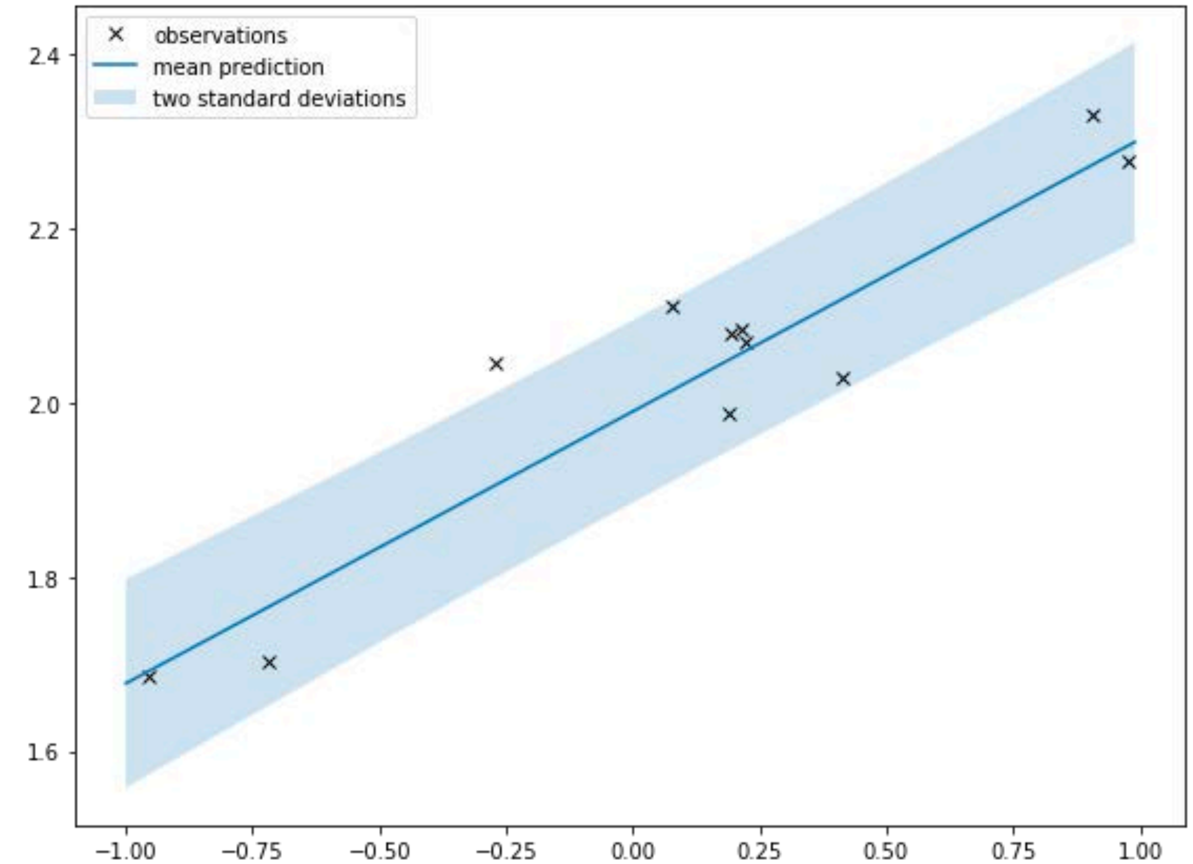


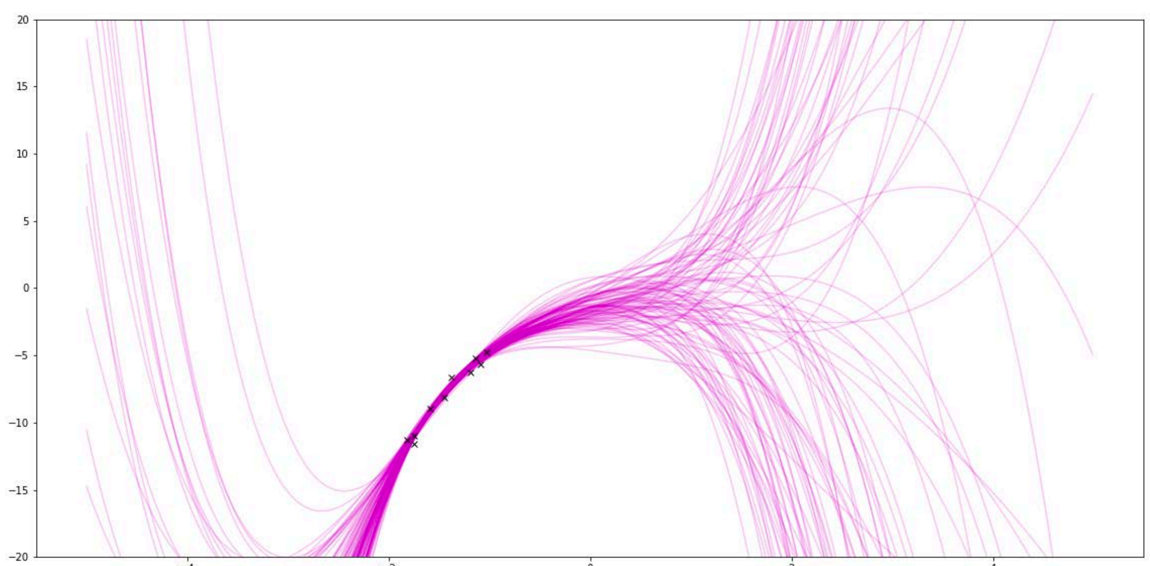
Figure 3.1 Examples of basis functions, showing polynomials on the left, Gaussians of the form (3.4) in the centre, and sigmoidal of the form (3.5) on the right.

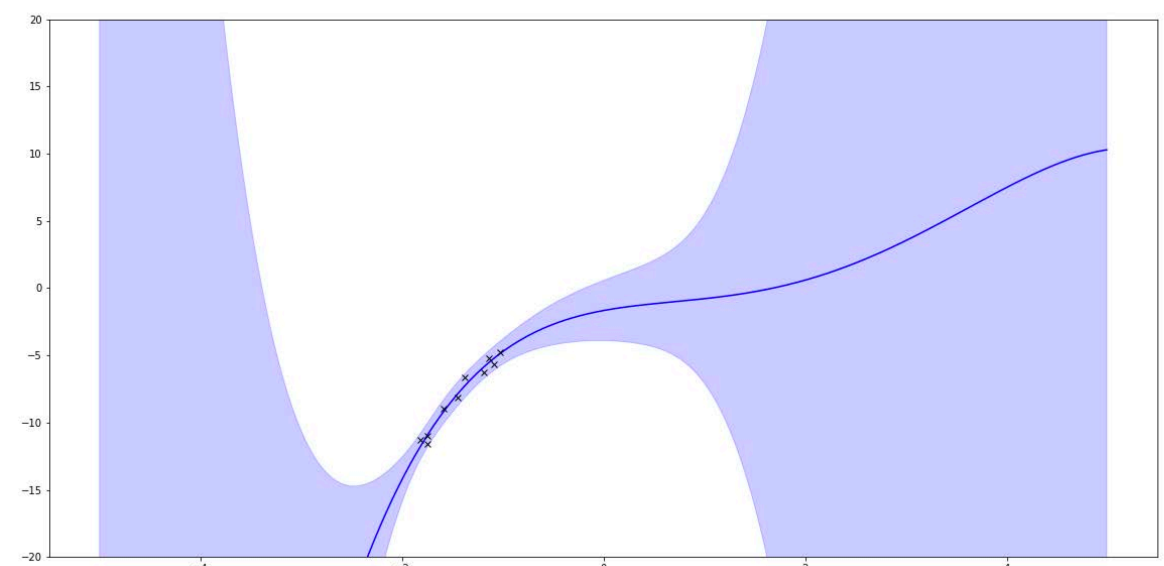
Bayesian linear regression with basis functions





Nonlinear functions can be approximating using basis functions (or features)





$$\boldsymbol{y} = w^T \phi(\boldsymbol{x}) + \epsilon$$

Geometrical interpretation

Figure 3.2 Geometrical interpretation of the least-squares solution, in an N-dimensional space whose axes are the values of $t_1, \ldots, t_N$. The least-squares regression function is obtained by finding the orthogonal projection of the data vector $\mathbf{t}$ onto the subspace spanned by the basis functions $\phi_j(\mathbf{x})$ in which each basis function is viewed as a vector $\boldsymbol{\varphi}_j$ of length N with elements $\phi_j(\mathbf{x}_n)$.