COMPSCI 589
Lecture 13: Probabilistic Regression Models

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

The Regression Task

Definition: The Regression Task

Given a feature vector $\mathbf{x} \in \mathbb{R}^D$, predict its corresponding output value $y \in \mathbb{R}$.

The Regression Learning Problem

Definition: Regression Learning Problem

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1 : N\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ is a feature vector and $y_i \in \mathbb{R}$ is the output, learn a function $f : \mathbb{R}^D \to \mathbb{R}$ that accurately predicts y for any feature vector $\mathbf{x}$.

The Probabilistic Regression Task

Definition: The Probabilistic Regression Task

Given a feature vector $\mathbf{x} \in \mathbb{R}^D$, predict a conditional probability density function $p(\mathbf{Y}|\mathbf{X} = \mathbf{x})$ over the output $y \in \mathbb{R}$.

The Probabilistic Regression Learning Problem

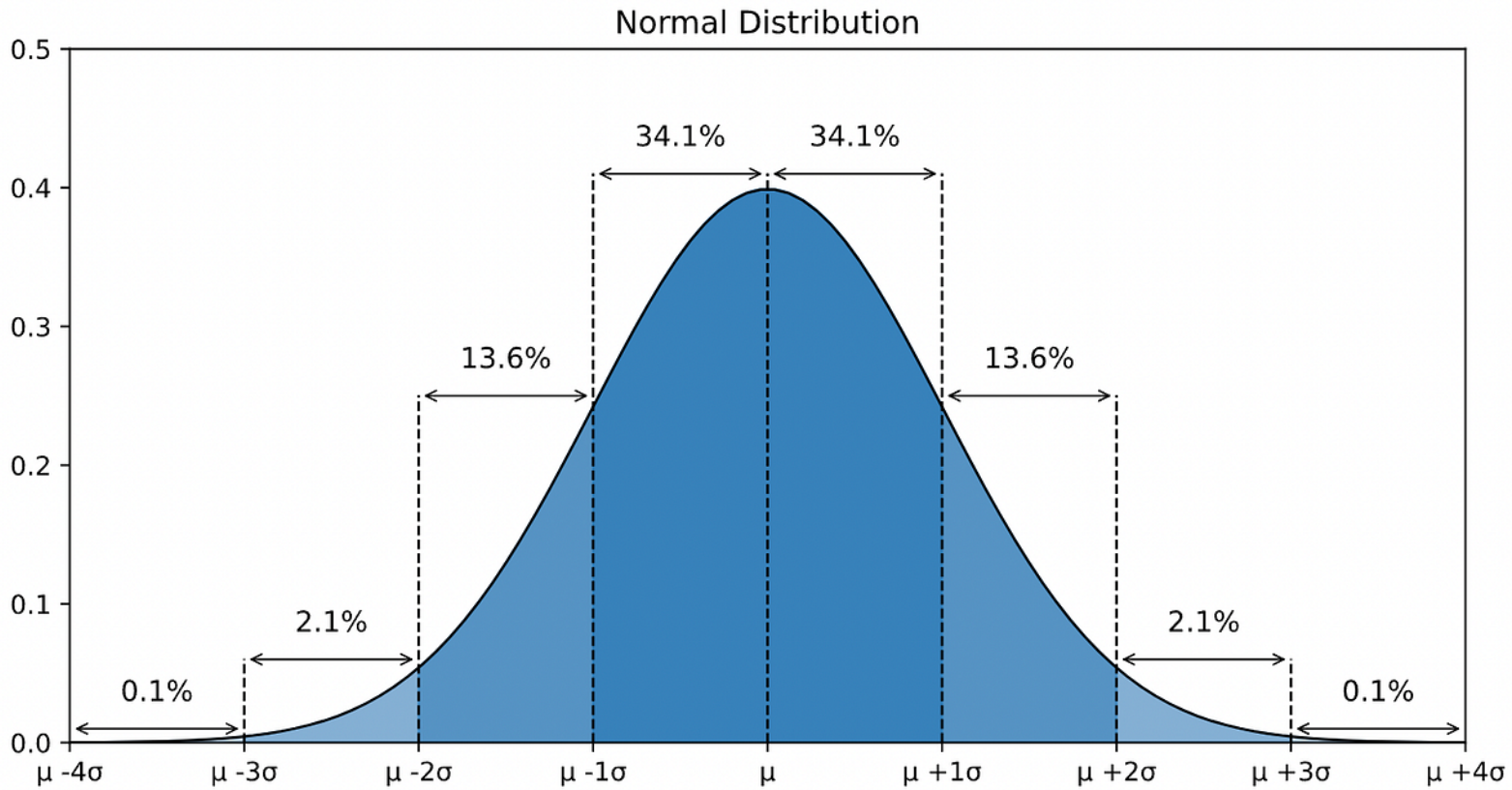
Definition: Probabilistic Regression Learning Problem

Given a data set of example pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1 : N\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ is a feature vector and $y_i \in \mathbb{R}$ is the output, output a conditional probability density function p that places high probability density $p(\mathbf{Y} = y|\mathbf{X} = \mathbf{x})$ on the true output y for any input $\mathbf{x}$.

Relevance of Probabilistic Regression

- Why do we need to model the uncertainty in a regression problem?
- If we are using the output of the regression model to make decisions, we usually want a measure of confidence.
- For example, a deterministic temperature prediction model might predict that the overnight low will 5 degrees above freezing.
- A probabilistic temperature prediction model might output a temperature density prediction with same mean that shows a 5% chance that the temperature is below freezing and a 95% chance that the temperature is above freezing.
- Or, it might output a temperature density prediction with same mean, but a 45% chance that the temperature is below freezing and a 55% chance that the temperature is above freezing.

Relevance of Probabilistic Regression



Probabilistic Linear Regression

- Probabilistic linear regression models the conditional mean of $p(\mathbf{Y} = y|\mathbf{X} = \mathbf{x})$ as a linear regression function: $f_{Lin}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$.
- It models the uncertainty around the conditional mean function $f_{Lin}(\mathbf{x})$ using a normal distribution with constant standard deviation σ.

$$p_\theta(Y = y|\mathbf{X} = \mathbf{x}) = \mathcal{N}(y; f_{Lin}(\mathbf{x}), \sigma^2)$$
$$= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - f_{Lin}(\mathbf{x}))^2\right)$$

Probabilistic Linear Regression: Learning

- To learn the probabilistic regression model parameters $\theta = [\mathbf{w}; b; \sigma]$, we minimize the negative average log likelihood:

$$nall(\theta, \mathcal{D}_{tr}) = -\frac{1}{N_{tr}} \sum_{i=1}^{N} \log p_\theta(Y = y_i | \mathbf{X} = \mathbf{x}_i)$$

$$\hat{\theta} = \arg\min_\theta \; nall(\theta, \mathcal{D}_{tr})$$

Probabilistic Linear Regression: Learning

- The solution to this learning problem turns out to be to fit the weights $\mathbf{w}$ and the bias b using the closed-form OLS linear regression solution.
- Once we have learned $\hat{\mathbf{w}}$ and $\hat{b}$, there is a simple, closed-form rule for the optimal standard deviation parmmeter value:

$$\hat{\sigma} = \sqrt{\frac{1}{N_{tr}} \sum_{i=1}^{N} (y_i - f_{Lin}(\mathbf{x}))^2}$$

Probabilistic Neural Networks

- Neural Network Regression models can also easily be made probabilistic by using a NN regression function $f_{NN}(\mathbf{x})$ as the conditional mean of a normal distribution and choosing an appropriate standard deviation value σ:

$$p_\sigma(Y = y|\mathbf{X} = \mathbf{x}) = \mathcal{N}(y; f_{NN}(\mathbf{x}), \sigma^2)$$
$$= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - f_{NN}(\mathbf{x}))^2\right)$$

- We can do this using any architecture as the neural network regressor $f_{NN}(\mathbf{x})$ from a basic MLP, to a CNN, RNN, or even transformer-based LLM.

Probabilistic NN Regression: Learning

- To learn the probabilistic NN regression model parameters $\theta = [\phi; \sigma]$, we minimize the negative average log likelihood:

$$nall(\theta, \mathcal{D}_{tr}) = -\frac{1}{N_{tr}} \sum_{i=1}^{N} \log p_\theta(Y = y_i | \mathbf{X} = \mathbf{x}_i)$$

$$\hat{\theta} = \arg\min_\theta \; nall(\theta, \mathcal{D}_{tr})$$

- The solution again simplifies to minimizing the squared error for the neural network regression function $f_{NN}(\mathbf{x})$, followed by estimating the standard deviation parameter based on the residuals:

$$\hat{\sigma} = \sqrt{\frac{1}{N_{tr}} \sum_{i=1}^{N} (y_i - f_{NN}(\mathbf{x}))^2}$$

Probabilistic KNN

- KNN can made probabilistic by using the KNN regression function $f_{KNN}(\mathbf{x})$ as the conditional mean of a normal distribution and choosing an appropriate standard deviation value σ:

$$p_\sigma(Y = y|\mathbf{X} = \mathbf{x}) = \mathcal{N}(y; f_{KNN}(\mathbf{x}), \sigma^2)$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - f_{KNN}(\mathbf{x}))^2\right)$$

$$f_{KNN}(\mathbf{x}) = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\mathbf{x})} y_i$$

Probabilistic KNN: Learning Sigma

- The probabilistic KNN function has a single parameter σ. We can learn this parameter by optimizing the negative average log likelihood:

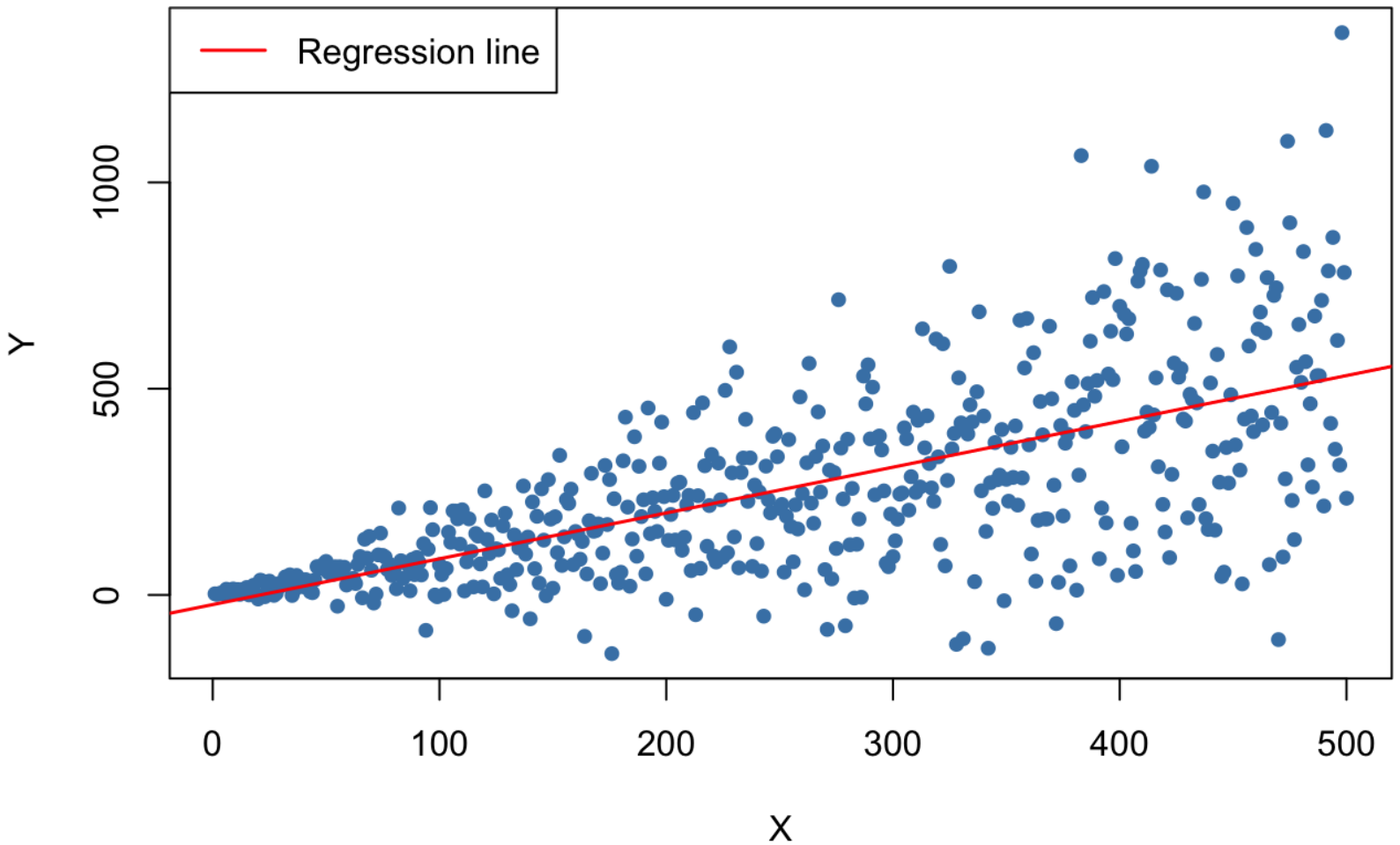
$$nall(\sigma, \mathcal{D}_{tr}) = -\frac{1}{N_{tr}} \sum_{i=1}^{N} \log p_{\sigma}(Y = y_i | \mathbf{X} = \mathbf{x}_i)$$

- This optimization problem has a simple, closed-form solution:

$$\hat{\sigma} = \sqrt{\frac{1}{N_{tr}} \sum_{i=1}^{N} (y_i - f_{KNN}(\mathbf{x}))^2}$$

Beyond Constant Spread

What if the uncertainty around the conditional mean is not constant?



Gaussian Process Regression

- Gaussian process regression is a powerful non-parametric regression model closely related to basis expansions and KNN that has non-constant uncertainty.
- The model is driven by the choice of a *covariance function* $\mathcal{K}$ that for any pair of inputs $\mathbf{x}$ and $\mathbf{x}'$ outputs a similarity score $\mathcal{K}(\mathbf{x}, \mathbf{x}')$.
- The covariance functions used have the further property that when given a collection of inputs $\mathbf{x}_1, \ldots, \mathbf{x}_N$, the $N \times N$ *kernel matrix* $\mathbf{K}$ with $\mathbf{K}_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$ is always strictly positive definite.

Gaussian Process Regression: Model

- Interestingly, the Gaussian Process regression model is able to produce complex, probabilistic regression functions with non-constant uncertainty using a minimal number of parameters (2-3).
- The predictive distribution is conditionally normal with a mean function and a variance function.

$$p_\sigma(Y = y|\mathbf{X} = \mathbf{x}) = \mathcal{N}(y; f(\mathbf{x}), v(\mathbf{x}))$$

- However, these functions depend directly on the training data set via the kernel function, similar to KNN.

Gaussian Process Regression: Prediction

- Let $\mathbf{K}_{tr}$ be the $N \times N$ kernel matrix between the feature vectors in the training data set $\mathcal{D}_{tr}$.
- Let $\mathbf{K}_*$ be the $1 \times N$ kernel matrix between the input $\mathbf{x}$ we are making a prediction for and all of the training set feature vectors.
- Let $\mathbf{Y}$ be a $N \times 1$ matrix of output values from the training set.
- The mean and variance functions are defined as:

$$f(\mathbf{x}) = \mathbf{K}_* \left(\mathbf{K}_{tr} + \gamma^2 I\right)^{-1} \mathbf{Y}$$
$$v(\mathbf{x}) = \mathcal{K}(\mathbf{x}, \mathbf{x}) - \mathbf{K}_* \left(\mathbf{K}_{tr} + \gamma^2 I\right)^{-1} \mathbf{K}_*^T$$

Gaussian Process Regression: Covariance Functions

- **Linear covariance:** Captures simple linear relationships between inputs.

$$\mathcal{K}_{Lin}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}' + c$$

- **Exponential covariance:** Measures similarity that decays exponentially with distance.

$$\mathcal{K}_{exp}(\mathbf{x}, \mathbf{x}') = \beta \exp\left(-\alpha \|\mathbf{x} - \mathbf{x}'\|_2^2\right)$$

- **Polynomial covariance:** Equivalent to linear covariance function applied following degree B polynomial basis expansion.

$$\mathcal{K}_{Pol}(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + c)^B$$

Variable Uncertainty in Regression

- A probabilistic regression model that asserts that the uncertainty in the regression outputs does not vary with the input $\mathbf{x}$ is called *homoscedastic*.
- A probabilistic regression model that allows the uncertainty in the regression outputs to vary with the input $\mathbf{x}$ is called *heteroscedastic*.
- Aside: “scedastic" is derived from the Greek word “skedastikos," which means “able to be scattered" or “dispersed."

Heteroscedastic KNN

- We can make KNN model variable uncertainty by augmenting the model so that the standard deviation used in the conditional normal model also depends on the input $\sigma(\mathbf{x})$.
- We can use a separate KNN-based standard deviation estimator to model $\sigma(\mathbf{x})$:

$$\sigma(\mathbf{x}) = \sqrt{\frac{1}{K} \sum_{i \in \mathcal{N}_K} (y_i - f_{KNN}(\mathbf{x}))^2}$$

- This will only make sense for reasonably large K. For small K, it will overfit badly.

Heteroscedastic Linear and NN Regression

- To make linear regression or neural network regression model variable uncertainty, we can also stay within the conditional normal family of models with a parametric (linear or non-linear) conditional mean function $f(\mathbf{x})$.
- As with Heteroscedastic KNN, we need to make the standard deviation input-dependent via a standard deviation function $\sigma(\mathbf{x})$.
- However, the standard deviation needs to be non-negative. If we want a parametric function for $\sigma(\mathbf{x})$, it can not be a linear function.

Heteroscedastic Linear and NN Regression

- The simplest approach for $\sigma(\mathbf{x})$ is often to model it as:

$$\sigma(\mathbf{x}) = \exp(g(\mathbf{x}))$$

  where $g(\mathbf{x})$ produces arbitrary (positive or negative) real-valued outputs, and the application of the exponential function ensures $\sigma(\mathbf{x}) \geq 0$ for any input $\mathbf{x}$.
- This allows us to make $f(\mathbf{x})$ linear or non-linear as desired, and to separately select a parametric form for $\sigma(\mathbf{x})$ via the choice of an architecture for $g(\mathbf{x})$.

Heteroscedastic Linear and NN Regression

- The final model is given below. We can jointly learn the parameters of the mean function $f(\mathbf{x})$ and the standard deviation function $\sigma\mathbf{x}$ using nall minimization.

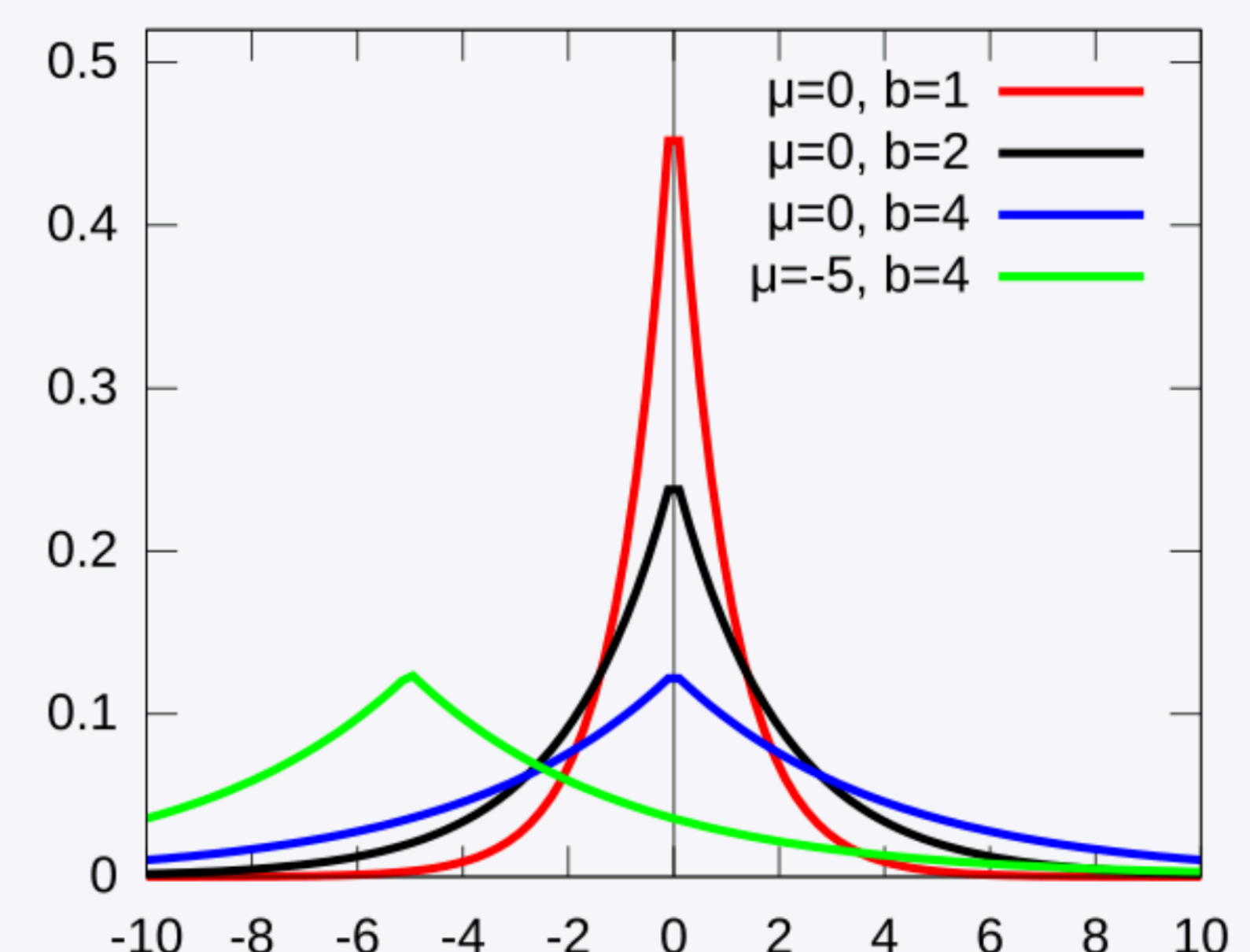
$$p_\sigma(Y = y|\mathbf{X} = \mathbf{x}) = \mathcal{N}(y; f(\mathbf{x}), \sigma(\mathbf{x})^2)$$

- One issue with learning is that allowing the uncertainty to be input-dependent often results in many more local optima when learning.
- A good strategy for learning is often to learn $f(\mathbf{x})$ with $\sigma(\mathbf{x})$ constrained to be constant until the mean function is providing a good fit, then to optimize the parameters of $\sigma(\mathbf{x})$.

Beyond The Normal Distribution

- The basic approach we have described so far models the conditional mean of the predictive distribution using a standard, non-probabilistic regression function $f(\mathbf{x})$ fit using squared error.
- It then models the uncertainty around the conditional mean with a Normal distribution with a fixed standard deviation.
- This approach can be generalized to other types of conditional probability density functions that can be parameterized in terms of a mean function and a constant spread parameter.
- The learning problem does not necessarily decompose as in the conditional normal case, but so long as the nall is (sub) differentiable, we can learn the model parameters via automatic differentiation.

Laplace Distribution



Gamma Distribution