Supervised Learning

Muchang Bahng

Summer 2023

Contents

From Bishop's "Pattern Recognition and Machine Learning" with guidance from Enmao Diao.

1 Introduction

Let us go over some notation first. Our input space of input data, aka **explanatory variables**, is usually almost always $\mathcal{X} = \mathbb{R}^d$, where D represents the dimension of our input space. N usually denotes the number of data points we have, and so we would have data of the form

$$\begin{aligned}\mathbf{x}^{(1)} &= \left(x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \dots, x_d^{(1)}\right)\\ \mathbf{x}^{(2)} &= \left(x_1^{(2)}, x_2^{(2)}, x_3^{(2)}, \dots, x_d^{(2)}\right)\\ \dots &= \dots\\ \mathbf{x}^{(N)} &= \left(x_1^{(N)}, x_2^{(N)}, x_3^{(N)}, \dots, x_d^{(nN)}\right)\end{aligned}$$

If we just stack the $\mathbf{x}^{(n)}$'s on top of each other, then we get a $N \times D$ **design matrix** $\mathbf{\Phi}$. In **supervised learning**, these input vectors $\mathbf{x}^{(n)}$ would also have a **target vector** $y^{(n)} \in \mathcal{Y}$ labeling them, and our goal is to find a **hypothesis function** $h : \mathcal{X} \to \mathcal{Y}$ that best predicts what label $\hat{y}$ we should give a new input $\hat{x}$. If $\mathbf{Y}$ is discrete, then estimating h is known as a **classification** problem and if $\mathbf{Y}$ is continuous, then this is a **regression** problem.

Normally, we would use some algorithm to directly or recursively build this function h, which is known as the **training phase**. Once the model is trained it can then determine the label of new data, which is the **test set**. The ability to label the new data correctly is known as **generalization**. Often more than not, the data is **preprocessed** to make it easier to learn. This can be done in several ways:

1. We may project the data onto a lower dimensional space or perform **feature extraction** if D is too big.
2. We may embed the data in a higher dimensional space to take advantage of certain geometric properties of the data.
3. We may translate and scale images so that they are all the same size.

In **unsupervised learning problems**, we do not have labels on our input data, and so our goal is a bit different. We can discover groups of similar examples within the data, known as **clustering**, or to determine the distribution of data within the input space, known as **density estimation**, or to project the data from a high-dimensional space to lower dimensions, known as **dimensionality-reduction**.

We usually assume that these $x^{(n)}$'s are all iid random variables that come from an unknown distribution X over $\mathbb{R}^N$ and the $y^{(i)}$'s come from an unknown distribution Y over $\mathbb{R}$. Ideally, we would like to know the joint distribution $X \times Y$ to observe the relationship between X and Y, but we don't. Regression is basically trying to estimate the function $\mathbb{E}[Y \mid X] = g(X)$, where $g : \mathbb{R}^n \to \mathbb{R}$ is some function.

1.1 Polynomial Regression

We start off with a simple example of regression to introduce concepts that will come up a lot, including loss/utility functions, overfitting, regularization, and model selection. Say that $\mathcal{X} = \mathbb{R}$ and we have the input-output data $\{x^{(i)}, y^{(i)}\}_{i=1}^N$. Now, let's fix a function $h_{\mathbf{w}}$, indexed by a parameter vector $\mathbf{w}$, of form

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M$$

Even though h is a nonlinear function of x, it is a linear function of $\mathbf{w}$, and so this is called linear regression. We would like to fit $h_{\mathbf{w}}(x)$ so that it best predicts y according to the data, and one way we can do this is to minimizing an **error function** $E(\mathbf{w})$ that determines how bad the function h is at fitting the data. One common choice is the **sum-of-squares** error function

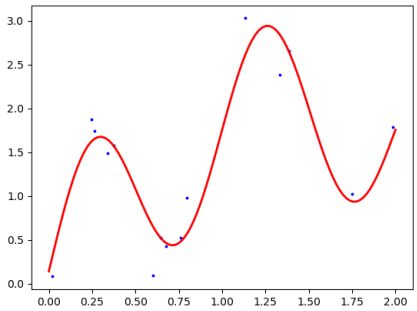
$$E(\mathbf{w}) \coloneqq \frac{1}{2} \sum_{n=1}^{N} \left(h_{\mathbf{w}}(x^{(n)}) - y^{(n)}\right)^2$$

which basically squares the residuals and sums them up. Another choice is simply summing up the $L1$ differences of the residuals.

$$E(\mathbf{w}) \coloneqq \frac{1}{2}\sum_{n=1}^{N} |h_{\mathbf{w}}(x^{(n)} - y^{(n)}|$$

and the motivation for these two cost functions will be explained later. Now let us have a practical example in which we generate a bunch of data points on the interval $[0, 1]$ of the function $f(x) = \sin(2\pi x) + 2\cos(x - 1.5)$, with an additional Gaussian noise $N(0, 0.3)$. This can be generated using `numpy` with the following code.

```
np.random.seed(60)
N = 15
X = np.random.uniform(0, 2, N)
Y = np.sin(2 * np.pi * X) + 2 * np.cos(X - 1.5) + np.random.normal(0, 0.3, N)

plt.scatter(X, Y, c="b", s=3, label="Dataset")
x_ran = np.linspace(0, 2, 100)
y_ran = np.sin(2 * np.pi * x_ran) + 2 * np.cos( x_ran - 1.5)
plt.plot(x_ran, y_ran, c="r", linewidth=3, label="True Function")
plt.show()
```



It looks like a fifth degree polynomial fits this data the best, but how would we know what degree we should do at all? Let's just try fitting for different degree polynomials. In order to do this, we use the `PolynomialFeatures` class, which essentially turns the input data $\{x^{(n)}\}$ into vectors

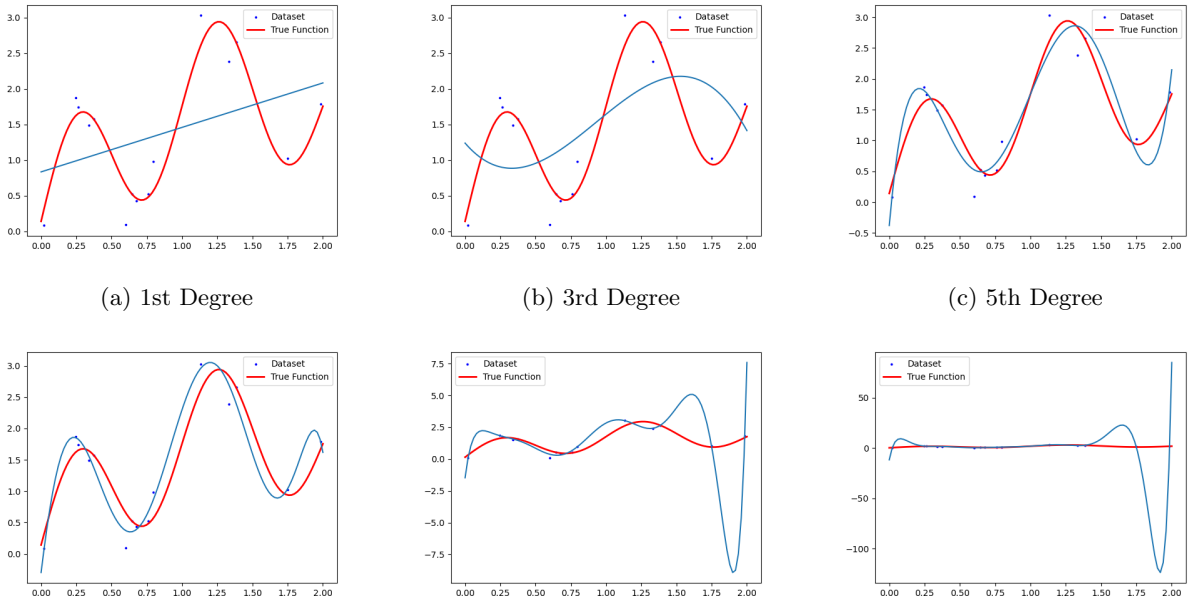
$$x^{(n)} \mapsto \left(1, (x^{(n)})^1, (x^{(n)})^2, \dots, (x^{(n)})^M\right)$$

and then we run linear regression on these vectors using the `LinearRegression` class.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

X_poly = PolynomialFeatures(degree = 3).fit_transform(X)
model = LinearRegression().fit(X_poly, Y)
```

Running this over degrees in `np.arange(1, 13, 2)` gives the plots.

We already know that the 5th degree approximation is most optimal, and the lower degree ones are **underfitting** the data, while the higher degree ones are **overfitting**. What overfitting essentially means is that our model is too complex to the point where it starts to fit to the *noise* of the data. What we usually have is a **training set** that allows us to train the model, and then to check its performance we have a **test set**. We can either take our original dataset $\{(x^{(n)}, y^{(n)}\}$ and partition it into a training and testing set, which is called **cross-validation**. Though we haven't made a testing set, since we know the true model let



(a) 1st Degree (b) 3rd Degree (c) 5th Degree

(d) 7th Degree (e) 9th Degree (f) 11th Degree

us just generate more data and use that as our testing set. For each model, we can calculate the optimal $\mathbf{w}$, which we will denote $\mathbf{W}^*$, and we will compute the **root mean square** error of our function $h_{\mathbf{w}^*}$ defined

$$E_{RMS} = \sqrt{2E(\mathbf{w}^*)/N}$$

where division of N allows us to compare different sizes of datasets on equal footing, and the square root ensures that this is scaled correctly. Let us see how well these different order models perform on a separate set of data generated by the same function with Gaussian noise. In 2, we can see that the RMS decreases

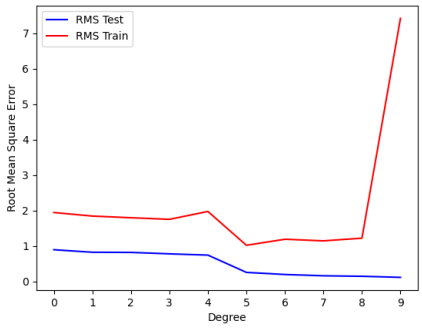
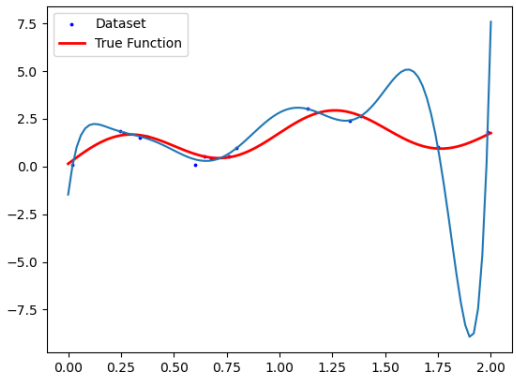


Figure 2: Degree of Polynomial vs RMS on Training and Testing Sets

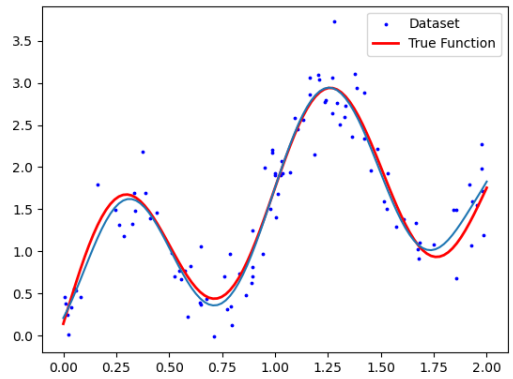
monotonically on the training error as more complex functions become more fine-tuned to the data. However, when we have a 9th degree polynomial the RMS for the testing set dramatically increases, meaning that this model does not predict the testing set well, and performance drops.

This overfitting problem is reduced if we have more training data to work with. That is, if we were to fit a 9th degree polynomial on a training set of not $N = 15$, but $N = 100$ data points, then we can see that this gives a much better fit.

So far, there are 2 ways to reduce the overfitting problem, either the decrease the model complexity or get more training data. A third way is to use a **regularizing** term, which adds a penalty term to the error



(a) $M = 9, N = 15$



(b) $M = 9, N = 100$

Figure 3: Increasing the number of data points helps the overfitting problem. Now, we can afford to fit a 9th degree polynomial with reasonable accuracy.

function in order to discourage the coefficients from reaching large values. The two most popular ones are **ridge regression**

$$E(\mathbf{w}) := \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}(x^{(n)}) - y^{(n)}\right)^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

which involves using the L2 norm of $\mathbf{w}$ and **LASSO regression**

$$E(\mathbf{w}) := \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}(x^{(n)}) - y^{(n)}\right)^2 + \frac{\lambda}{2}|\mathbf{w}|_1$$

which uses the L1 norm. These models are known as **shrinkage models**. We can add this term for the degree $M = 9$ polynomial.

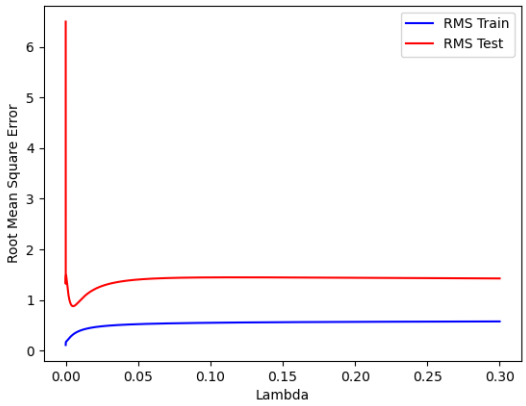


Figure 4: Even with a slight increase in the regularization term λ, the RMS error on the testing set heavily decreases.

1.1.1 Matrix Formulation

Ordinary least squares linear regression involves taking our data

$$\mathbf{x}^{(n)} = \left(x_1^{(n)}, x_2^{(n)}, x_3^{(n)}, \dots, x_d^{(n)}\right)$$

and minimizing the cost function

$$E(\mathbf{w}) \coloneqq \frac{1}{2}\sum_{n=1}^{N}\left(h_{\mathbf{w}}(x^{(n)}) - y^{(n)}\right)^2$$

where $h_{\mathbf{w}}(x)$ is linear in $\mathbf{w}$, of form $h_{\mathbf{w}}(x^{(n)}) = \sum_{j=0}^{M} w_j x_j^{(n)}$. Again, polynomial fitting is simply linear regression on the coefficients. Let us minimize this by taking its partials equal to 0, which after some computation gives

$$\begin{aligned}\frac{\partial}{\partial w_i}E(\mathbf{w}) &= \sum_{n=1}^{N}\left[\left(\sum_{j=0}^{M} w_j x_j^{(n)}\right) - y^{(n)}\right]x_i^{(n)} \\ &= \sum_{n=1}^{N}\left[\left(\sum_{j=0}^{M} w_j x_j^{(n)} x_i^{(n)}\right) - y^{(n)}x_i^{(n)}\right]0\end{aligned}$$

which implies that

$$\sum_{j=0}^{M} w_j \sum_{n=1}^{N} x_j^{(n)} x_i^{(n)} = \sum_{n=1}^{N} y^{(n)} x_i^{(n)}$$

Now compare this to the linear equation $\mathbf{\Phi w} = \mathbf{y}$, where $\mathbf{\Phi}$ is the $N \times D$ design matrix (in linear regression, we have $M = D$). The least squares solution to this possibly overdetermined system is $\mathbf{\Phi}^T\mathbf{\Phi w} = \mathbf{\Phi}^T\mathbf{y}$. Both sides are $d \times 1$ column vectors, so let us compute the ith element of each side. On the RHS it is

$$\left(x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(N)}\right)\begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \sum_{n=1}^{N} y^{(n)} x_i^{(n)}$$

and on the LHS, to compute the ijth element of $\mathbf{\Phi}^T\mathbf{\Phi}$, it is

$$(\mathbf{\Phi}^T\mathbf{\Phi})_{ij} = \sum_{n=1}^{N} x_i^{(n)} x_j^{(n)}$$

and now to compute the ith element of this product we must dot product $\mathbf{w}$ with the ith row, which gives us

$$\sum_{j=1}^{M} w_j \cdot (\mathbf{\Phi}^T\mathbf{\Phi})_{ij} = \sum_{j=0}^{M} w_j \sum_{n=1}^{N} x_j^{(n)} x_i^{(n)}$$

and so the familiar least squares formula we see in linear algebra is precisely the solution for the minimum of the sum of squares error function!

Now if we wanted to minimize the ridge regularized least squares, taking the partials will lead to getting

$$\left[\sum_{j=0}^{M} w_j \sum_{n=1}^{N} x_j^{(n)} x_i^{(n)}\right] + \lambda w_i = \sum_{n=1}^{N} y^{(n)} x_i^{(n)}$$

which would correspond to the matrix equation

$$(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I})\mathbf{w} = \mathbf{\Phi}^T\mathbf{y}$$

1.2 Statistics and Bayesian Probability

1.2.1 Sampling Bias and Bessel's Correction Factor

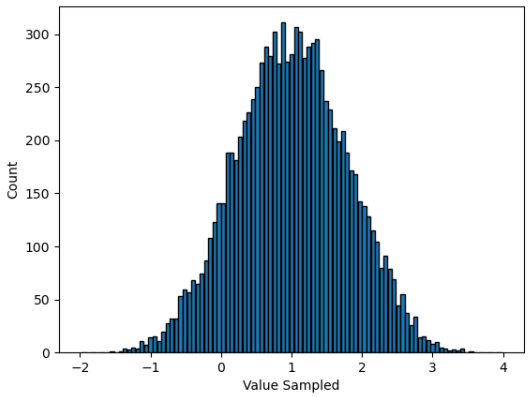
A statistics problem will start with some true distribution X defined over $\mathcal{X}$, which is known as the **population**. The true mean and variance is denoted $\mu = \mathbb{E}[X]$ and $\sigma^2 = \text{Var}[X]$, and they are almost always not known. From this population,we take N samples $x^{(1)}, \dots, x^{(N)}$ from it, which are just iid versions of X. We can interpret this as sampling from the random vector $\mathbf{x} = (x^{(1)}, \dots, x^{(N)}) \in \mathcal{X}^N$. The sample mean

$$\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x^{(n)}$$

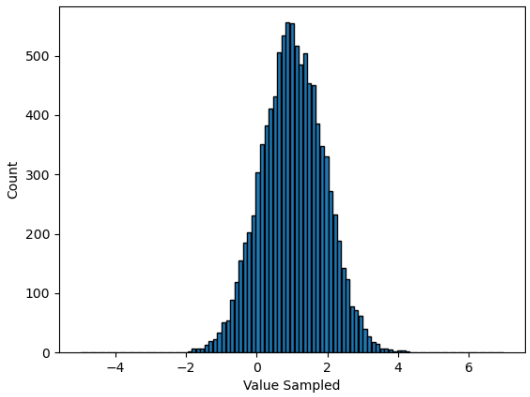
is a random variable and also an unbiased estimator since

$$\mathbb{E}[\bar{x}] = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[x^{(n)}] = \mu$$

and therefore, if we keep sampling $\bar{x}$ from $\mathcal{X}^N$, this distribution's mean will eventually converge to the true mean μ. We can see this by sampling sets of $N = 5$ points from the following distributions and calculating their sample mean. We do this 10000 times for a $N(1, 4)$ distribution and a $U(-2, 4)$ distribution. By CLT, we expect this distribution to look normal, regardless of which distribution it came from, as shown in Figure 9.



(a) 10000 samples from Uniform, Mean Sample Mean = 0.998



(b) 10000 samples from Gaussian, Mean Sample Mean = 1.018

Figure 5: The expected sample mean coincides empirically with the true mean.

Let us define the sample variance to be the random variable

$$S^2 = \frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})^2$$

Note that the sample variance uses the *sample mean*, rather than the true mean, and this leads to a bias in this estimation. Let us take its expectation:

$$\mathbb{E}[S^2] = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[(x_n - \bar{x})^2] = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbb{E}[x_n^2] + \mathbb{E}[\bar{x}^2] - 2\mathbb{E}[x_n \bar{x}] \right)$$

The first term evaluates to $\mathbb{E}[x_n^2] = \mu^2 + \sigma^2$. The second term

$$\mathbb{E}[\bar{x}^2] = \mathbb{E}\left[\frac{1}{N^2}\left(\sum_{n=1}^{N} x_n\right)^2\right] = \frac{1}{N^2}\left[\sum_{n=1}^{N}\mathbb{E}[x_n^2] + \sum_{i,j;i\neq j}\mathbb{E}[x_i]\mathbb{E}[x_j]\right]$$
$$= \frac{1}{N^2}\big(N(\mu^2+\sigma^2) + N(N-1)\mu^2\big) = \mu^2 + \frac{1}{N}\sigma^2$$

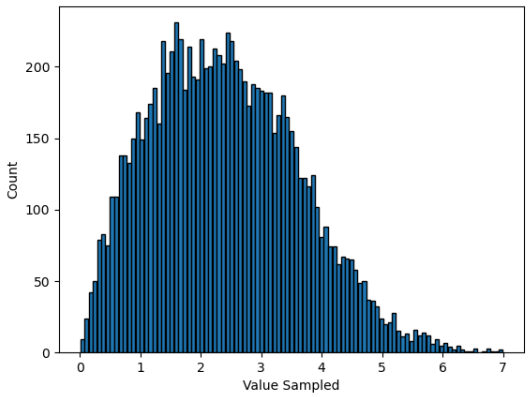
The third term evaluates

$$\mathbb{E}\left[x_n \frac{1}{N}\sum_{j=1}^{N} x_j\right] = \frac{1}{N}\mathbb{E}\left[x_n^2 + \sum_{j\neq n} x_j x_n\right]$$
$$= \frac{1}{N}\left[(\mu^2+\sigma^2) + \sum_{j\neq n}\mu^2 = \mu^2\right] = \mu^2 + \frac{1}{N}\sigma^2$$

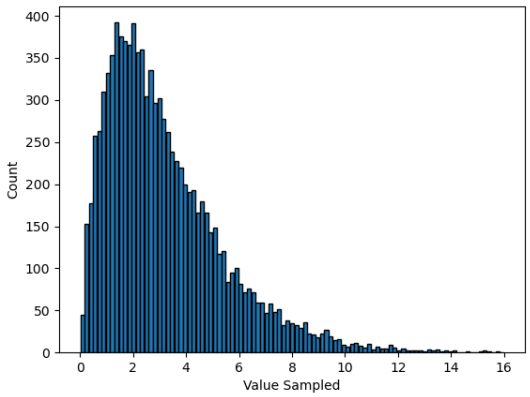
Substituting this above shows that the expectation of S^2 is not the true variance and this estimator is biased.

$$\mathbb{E}[S^2] = \frac{N-1}{N}\sigma^2$$

We can also see this empirically for the two distributions above. A $N(1,4)$ has a variance of 4 and the uniform $U(-2,4)$ has a variance of 3. Since $N = 5$, we should expect to see the sample variance to be $S^2 = 3.2$ and $S^2 = 2.4$, respectively, which we do in Figure 6.



(a) 10000 batches of 5 samples from Uniform, Mean Sample Variance = $2.384 \approx (4/5)\cdot 3$



(b) 10000 batches of 5 samples from Gaussian, Mean Sample Variance = $3.205 \approx (4/5)\cdot 4$

Figure 6: The expected sample mean coincides empirically with the true mean.

Therefore, the sample variance is often defined by scaling this by **Bessel's correction factor**, which leads to us divide the sum by $N-1$, not N.

$$S^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \bar{x})^2$$

If we actually knew the true mean and took the sample variance with this mean, then this would be unbiased.

$$\begin{aligned}\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}(x_n-\mu)^2\right] &= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[(x_n-\mu)^2] \\ &= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[x_n^2]+\mathbb{E}[\mu^2]-2\mathbb{E}[x_n\mu] \\ &= \frac{1}{N}\sum_{n=1}^{N}\mu^2+\sigma^2+\mu^2-2\mu^2=\sigma^2\end{aligned}$$

Intuitively, we can see that the sample variance underestimates the correction factor because we are taking the squared sum around the sample mean. To see closer, let $x_n - \mu$ be the difference between the sample and the true mean, and let $\mu - \bar{x}$ be the difference between the sample mean and the true mean. Then, we can decompose the variance as

$$\begin{aligned}\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n-\mu)^2 &= \frac{1}{N}\sum_{n=1}^{N}[(x_n-\bar{x})(\bar{x}-\mu)]^2 \\ &= \frac{1}{N}\sum_{n=1}^{N}(x_n-\bar{x})^2+(\bar{x}-\mu)^2+2(x_n-\bar{x})(\bar{x}-\mu) \\ &= \frac{1}{N}\sum_{n=1}^{N}(x_n-\bar{x})^2+\frac{1}{N}\sum_{n=1}^{N}(\bar{x}-\mu)^2\end{aligned}$$

where the last sum vanishes since we're summing over x_n. And so we have $\sigma^2 \geq S^2$, and equality is satisfied if and only if $\bar{x} = \mu$. Therefore, if the sample mean is further from the actual mean, we expect a greater deviation in the variance too.

1.2.2 Bayesian Probability

Now this book puts a heavy emphasis on Bayesian probabilistic models. For now, we will denote $p(X)$ to be the distribution of a random variable X. We capture our assumptions about the model parameter $\mathbf{w}$ with a prior distribution $p(\mathbf{w})$. Our likelihood $p(\mathcal{D} \mid \mathbf{w})$ is the conditional distribution of getting the data $\mathcal{D}$ from our model with parameter $\mathbf{w}$. Therefore, Bayes theorem is expressed

$$p(\mathbf{w} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \mathbf{w})\, p(\mathbf{w})}{p(\mathcal{D})}$$

The denominator $p(\mathcal{D})$ is a normalizing term equal to $\int p(\mathcal{D} \mid \mathbf{w})\, p(\mathbf{w})\, d\mathbf{w}$, and for high dimensional $\mathcal{W}$ it may not be feasible to compute this integral without monte carlo sampling. Therefore, we focus on the numerator terms and remember the rule

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

For clarification, $\mathcal{D}$ can represent different things depending on the problem:

1. In a density estimation problem, where we have a single dataset $\mathbf{X}$, $\mathcal{D} = \mathbf{X}$ since this data tells us information about which distribution it could come from.
2. In a regression problem, $\mathcal{D} = \mathbf{Y}$, that is, $\mathcal{D}$ will always be the output data, not the input data $\mathbf{X}$. We can think of the input data $\mathbf{X}$ as always being fixed, and it is upon observation of the *outputs* $\mathbf{Y}$ on these inputs that gives us information.

In both the frequentist and Bayesian settings, the likelihood $p(\mathcal{D} \mid \mathbf{w})$ plays a central role. In the frequentist setting, the process is divided into two steps:

1. We optimize $\mathbf{w}$ with some **estimator**, with a popular one being the **maximum likelihood estimator**. A popular estimator is **maximum likelihood**, which seeks to maximize $p(\mathcal{D} \mid \mathbf{w})$ w.r.t. $\mathbf{w}$.

2. We fix the optimized $\mathbf{w}^*$ and error bars on this estimate are obtained by considering the distribution of possible datasets $\mathcal{D}$. One approach is **bootstrapping**, which goes as follows. Given our original dataset $\mathbf{X} = \{x^{(1)}, \dots, x^{(N)}\}$, we can create a new dataset $\mathbf{X}'$ by sampling N points at random from $\mathbf{X}$, with replacement, so that some points in $\mathbf{X}$ may be replicated in $\mathbf{X}'$, whereas other points in $\mathbf{X}$ may be absent in $\mathbf{X}'$. This process is repeated L times to generate L different datasets. Then, we can look at the variability of prediction between the different bootstrap data sets.

In a Bayesian setting, there is only a single dataset $\mathcal{D}$ and the uncertainity in the parameters is expressed through a probability distribution over $\mathbf{w}$. It also includes prior knowledge naturally in the form of prior distributions.

1.3 Density Estimation

1.3.1 Frequentist Approach

As a start, let us have a dataset of observations $\mathbf{X} = \{x^{(1)}, \dots, x^{(n)}\}$ assuming that they are all iid from $X \sim N(0, 1)$ distribution. Since this is iid, we can look at the joint distribution X^N on $\mathbb{R}^N$ and get the likelihood of form

$$p(\mathbf{X} \mid \mu, \sigma^2) = \prod_{n=1}^{N} p_X(x^{(n)} \mid \mu, \sigma^2)$$

which in turn gives the log-likelihood as

$$\ln p(\mathbf{X} \mid \mu, \sigma^2) = -\frac{1}{2\sigma^2} \sum_{n=1}^{N} (x_n - \mu)^2 - \frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi)$$

This is a function of two variables, μ and σ^2 and we can optimize it to get the maximum likelihood estimates of

$$\mu_{ML} = \frac{1}{N} \sum_{n=1}^{N} x_n \text{ and } \sigma^2_{ML} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu_{ML})^2$$

However, as we saw in the previous section, the estimate for σ^2 is biased by a factor of $(N-1)/N$, and this is an intrinsic flaw in the frequentist approach.

1.3.2 Bayesian Approach

In the Bayesian approach, we want to model

$$p(x \mid \mathcal{D}) = \int p(x \mid \mathbf{w})\, p(\mathbf{w} \mid \mathcal{D})\, d\mathbf{w}$$

1.4 Regression with Regularization

1.4.1 Frequentist's Maximum Likelihood Approach

Now given the hypothesis function $h_{\mathbf{w}}$, researchers assume that the relationship between the X and Y values are captured by

$$Y = h_{\mathbf{w}}(X) + \epsilon$$

where ϵ is some residual noise, also a random variable. Researchers assume that this random variable has a nice form. One popular choice is that $\epsilon \sim N(0, \sigma^2)$ since if we assume that this error is due to a large number of weakly dependent unknown factors, then by CLT we can assume that their sum is Gaussian. But ultimately this is just another assumption. With this Gaussian assumption, we can assume that each input

output pair $(x^{(n)}, y^{(n)})$ is generated by form $y^{(n)} = h_{\mathbf{w}}(x^{(n)}) + \epsilon$ and so the conditional distribution of $y^{(n)}$ given $X^{(n)}$ is

$$Y \mid X = x^{(n)} \sim N(h_{\mathbf{w}}(x^{(n)}), \sigma^2)$$

and therefore, the probability of getting $y^{(n)}$ given $x^{(n)}$ is modeled by the conditional pdf

$$p_{Y|X=x^{(n)}}(y^{(n)}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{[y^{(n)} - h_{\mathbf{w}}(x^{(n)})]^2}{2\sigma^2} \right)$$

Extending this to the dataset $\mathcal{D} = \mathbf{Y}$ coming from the N-fold joint distribution of X, by independence this distribution is a multivariate Gaussian

$$Y^n \mid X^n = \mathbf{X} \sim N(h_{\mathbf{w}}(\mathbf{X}), \sigma^2 \mathbf{I})$$

where by abuse of notation, $h_{\mathbf{w}}(\mathbf{X})$ is $h_{\mathbf{w}}$ operated element-wise on the vector $\mathbf{X}$, and $\mathbf{I}$ is the $N \times N$ identity matrix. The pdf is

$$\begin{aligned} p_{Y^n|X^n=\mathbf{X}}(\mathbf{Y}) &= \prod_{n=1}^{N} p_{Y|X=x^{(n)}}(y^{(n)}) \\ &= \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{[y^{(n)} - h_{\mathbf{w}}(x^{(n)})]^2}{2\sigma^2} \right) \end{aligned}$$

The two parameters of interest here that we would like to maximize are $\mathbf{w}$ and σ^2. We can take the log of this function to maximize this, which gives us

$$\ell(\mathbf{w}, \sigma^2) = -\frac{1}{\sigma^2} E_D(\mathbf{w}) - \frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi)$$

and here we can see that maximizing the likelihood w.r.t. $\mathbf{w}$ is equal to minimizing the sum-of-squares error function $E_D(\mathbf{w}) = -\frac{1}{2} \sum_{n=1}^{N} [y^{(n)} - h_{\mathbf{w}}(x^{(n)})]^2$. Therefore, a maximum likelihood estimation under a Gaussian residual assumption implies minimization of the sum-of-squares error function! To maximize with respect to both $\mathbf{w}$ and σ^2, we can use the fact that this function is C^1 (continuously differentiable), and so we just need to find where the partials are 0. Ultimately, we can just optimize for $\mathbf{w}$ first and then solve for σ^2. If $h_{\mathbf{w}}$ was linear (not necessarily in $\mathbf{x}$, but with $\mathbf{w}$), then we can transform the x_d values, get the proper design matrix $\mathbf{\Phi}$, and compute

$$\mathbf{w}_{ML} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{Y}$$

If we add a ridge penalty term to get $E(\mathbf{w}) = E_D(\mathbf{w}) + \frac{\lambda}{2} ||\mathbf{w}||_2^2$, then this results in solving the matrix equation

$$\mathbf{w}_{ML} = (\mathbf{\Phi}^T \mathbf{\Phi} + \lambda \mathbf{I})^{-1} \mathbf{\Phi}^T \mathbf{Y}$$

With these optimized parameters, we have a **probabilistic model** in which given a new value $\hat{\mathbf{x}}$, we can predict the conditional distribution of $\hat{y}$ to be

$$p(y' \mid \hat{\mathbf{x}}, \mathbf{w}_{ML}, \sigma_{ML}^2) = N(\hat{y} \mid h_{\mathbf{w}_{ML}}(\mathbf{x}'), \sigma_{ML}^2)$$

1.4.2 Bayesian Approach

We will now demonstrate how having a normal $\alpha \mathbf{I}$ prior around the origin in a Bayesian setting is equivalent to having a ridge penalty of $\lambda = \sigma^2/\alpha^2$ in a frequentist setting. If we have a Gaussian prior of form

$$p(\mathbf{w} \mid \alpha^2) = N(\mathbf{w} \mid \mathbf{0}, \alpha^2 \mathbf{I}) = \left(\frac{1}{2\pi\alpha^2} \right)^{M/2} \exp\left(- \frac{1}{2\alpha^2} ||\mathbf{w}||_2^2 \right)$$

We can use Bayes rule to compute

$$\begin{aligned} p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}, \alpha^2, \sigma^2) &\propto p(\mathbf{Y} \mid \mathbf{w}, \mathbf{X}, \alpha^2, \sigma^2)\, p(\mathbf{w} \mid \mathbf{X}, \alpha^2, \sigma^2) \\ &= \left[\prod_{n=1}^{N} p(y^{(n)} \mid \mathbf{w}, \mathbf{x}^{(n)}, \alpha^2, \sigma^2) \right] p(\mathbf{w} \mid \mathbf{X}, \alpha^2, \sigma^2) \\ &= \left[\prod_{n=1}^{N} N(y^{(n)} \mid h_{\mathbf{w}}(\mathbf{x}^{(n)}), \sigma^2) \right] N(\mathbf{w} \mid \alpha^2) \\ &= \left[\prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{(y^{(n)} - h_{\mathbf{w}}(x^{(n)}))^2}{2\sigma^2} \right) \right] \cdot \left(\frac{1}{2\pi\alpha^2} \right)^{M/2} \exp\left(- \frac{1}{2\alpha^2} ||\mathbf{w}||_2^2 \right) \end{aligned}$$

and taking the negative logarithm gives us

$$\ell(\mathbf{w}) = \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left(y^{(n)} - h_{\mathbf{w}}(\mathbf{x}^{(n)}) \right)^2 + \frac{N}{2} \ln \sigma^2 + \frac{N}{2} \ln(2\pi) - \frac{M}{2} \ln(2\pi\alpha^2) + \frac{1}{2\alpha^2} ||\mathbf{w}||_2^2$$

taking out the constant terms relative to $\mathbf{w}$ and multiplying by $2\sigma^2$ (which doesn't affect optima) gives us the ridge penalized error with a penalty term of $\lambda = \sigma^2/\alpha^2$.

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(y^{(n)} - h_{\mathbf{w}}(\mathbf{x}^{(n)}) \right)^2 + \frac{\sigma^2}{\alpha^2} ||\mathbf{w}||_2^2$$

But minimizing this still gives a point estimate of $\mathbf{w}$, which is not the full Bayesian treatment. In a Bayesian setting, we are given the training data $(\mathbf{X}, \mathbf{Y})$ along with a new test point $\mathbf{x}'$ and want to evaluate the predictive distribution $p(y \mid \mathbf{x}', \mathbf{X}, \mathbf{Y})$. We can do this by integrating over $\mathbf{w}$.

$$\begin{aligned} p(y \mid \mathbf{x}', \mathbf{X}, \mathbf{Y}) &= \int p(y \mid \mathbf{x}', \mathbf{w}, \mathbf{X}, \mathbf{Y})\, p(\mathbf{w} \mid \mathbf{x}', \mathbf{X}, \mathbf{Y})\, d\mathbf{w} \\ &= \int p(y \mid \mathbf{x}', \mathbf{w})\, p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y})\, d\mathbf{w} \end{aligned}$$

where we have omitted the irrelevant variables, along with α^2 and σ^2 to simplify notation. By substituting the posterior $p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y})$ with a normalized version of our calculation above and by noting that

$$p(y \mid \mathbf{x}', \mathbf{w}) = N(y \mid h_{\mathbf{w}}(\mathbf{x}'), \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{\left(y - h_{\mathbf{w}}(\mathbf{x}') \right)^2}{2\sigma^2} \right)$$

Now this integral may or may not have a closed form, but if we consider the polynomial regression with the hypothesis function of form

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_{M-1} x^{M-1}$$

then this integral turns out to have a closed form solution given by

$$p(y \mid \mathbf{x}', \mathbf{X}, \mathbf{Y}) = N\big(y \mid m(x'), s^2(x')\big)$$

where

$$\begin{aligned} m(x') &= \frac{1}{\sigma^2} \boldsymbol{\phi}(x')^T \mathbf{S} \left(\sum_{n=1}^{N} \boldsymbol{\phi}(x^{(n)}) y^{(n)} \right) \\ s^2(x') &= \sigma^2 + \boldsymbol{\phi}(x')^T \mathbf{S} \boldsymbol{\phi}(x') \\ \mathbf{S}^{-1} &= \alpha^{-2} \mathbf{I} + \frac{1}{\sigma^2} \sum_{n=1}^{N} \boldsymbol{\phi}(x^{(n)}) \boldsymbol{\phi}(x')^T \end{aligned}$$

and $\boldsymbol{\phi}(x)$ is the vector of functions $\phi_i(x) = x^i$ from $i = 0, \ldots, M-1$.

1.5 Model Selection

In regression, since $\mathbb{E}[Y \mid X] = g(X)$ can be any $\mathcal{B}$-measurable function (where $\mathcal{B}$ is the borel σ-algebra of $\mathbb{R}^n$), there could be crazy functions out there that we might have to predict. This space may even be uncountably infinite dimensional and so we must try to fit an uncountable number of parameters. This is clearly too big, so we shrink our function space to something that is finite and more manageable. Here are some examples:

1. If we assume $g : \mathbb{R}^d \to \mathbb{R}$ to be linear, then g lives in the dual of $\mathbb{R}^d$, which we know to be d-dimensional.
2. If we assume g to be affine, then this just adds one more dimension.
3. If we assume $g : \mathbb{R} \to \mathbb{R}$ to be a kth degree polynomial, then g can be parameterized by a $k+1$ dimensional θ.

As of now, we are interested in functions of form where we can parameterize it with a vector θ, so we would write g_θ to emphasize that its parameters are θ. So how do we choose which form is the best? Well this is just an assumption that most researchers make, and this is called **model selection**.

Now let us talk about model selection. For now, a **model** $\mathcal{M}$ is a certain form of a function/distribution that we want to fit. We can think of it in regression as a subset of the function space of $\sigma(X)$-measurable functions or in density estimation, as a set of distributions. This can apply to density estimation, regression, and classification problems.

Example 1.1 (Polynomial Regression)**.** The number of terms M, i.e. the degree $M-1$ of the polynomial

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_{M-1} x^{M-1}$$

in polynomial regression gives us models with different complexities, where $\mathcal{M}_M$ determines the model with a $M-1$th degree polynomial.

Example 1.2. Suppose I have data sampled data $x^{(1)}, \ldots, x^{(N)}$ on age at death for N people from an unknown distribution X. Then, possible models that model the distribution are

1. $\mathcal{M}_1$: the exponential distribution $p(x \mid w) = we^{-wy}$ with parameter w.
2. $\mathcal{M}_2$: the gamma distribution $p(y \mid a, b) = (b^a/\Gamma(a))y^{a-1}e^{-by}$ with parameter $\sqsupseteq = (a, b)$.
3. $\mathcal{M}_3$: the log-normal distribution with $X \sim N(\mu, \sigma^2)$ where $\mathbf{w} = (\mu, \sigma^2)$.

Example 1.3. A mixture of Gaussians model

$$p(\mathbf{y}) = \sum_{m=1}^{M} \pi_m N(\mathbf{y} \mid \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$$

has submodels where we must determine the number of Gaussians M.

Now if we assume that the actual true distribution X or the true regressor $\mathbb{E}[Y \mid X]$ is contained within our model $\mathcal{M}$, then we say our model is **well-specified**. But since researchers have no idea what the data generating process is, so $\mathbb{E}[Y \mid X] \notin \mathcal{M}$. Hence there is the saying that saying that "all models are wrong," since we never know what the true data generating process is, and thus the quantity

$$\mathbb{E}[Y \mid X] - h^*_{\mathbf{w}}(X)$$

where $h^*_{\mathbf{w}}(X)$ is the optimized hypothesis functions within $\mathcal{M}$, will always be nonzero. How close we can get this quantity to 0 determines how useful the model is, and a misspecified model is fundamentally a convenient (or even necessary) assumption on the distribution underlying the data, which may only be a reasonable approximation.

1.5.1 Cross-Validation

Now we know that a more complex model (i.e. that captures a greater set of distributions) is not necessarily the best due to overfitting. Therefore, researchers perform **cross-validation** by taking the training set $(\mathcal{X}, \mathbf{Y})$. We divide it into S equal pieces

$$\bigcup_{s=1}^{S} D_s = (\mathcal{X}, \mathcal{Y})$$

Then, we train the model $\mathcal{M}$ on $S-1$ pieces of the data and then test it across the final piece, and do this S times for every test piece, averaging its perforance across all S test runs. Therefore, for every model $\mathcal{M}_k$, we must train it S times, for all K models, requiring KS training runs. If data is particularly scarce, we set $S = N$, called the **leave-one-out** technique. Then we just choose the model with the best average test performance.

To implement this in scikit-learn, we want to use the `train_test_split` class. We can also set a random state parameter to reproduce results.

```
from sklearn.model_selection import train_test_split

# Split into training (80\%) and test (20\%) data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
                                                    random_state=66)
```

However, this process requires a lot of training runs and therefore may be computationally infeasible. Therefore, various **information criterion** has been proposed to efficiently select a model.

1.5.2 Akaike Information Criterion

The **Akaike information criterion (AIC)** chooses the model for which the quantity

$$\ln p(\mathcal{D} \mid \mathbf{w}_{ML}) - M$$

is minimized, where M is the model complexity, i.e. the number of adjustable parameters in the model.

1.5.3 Bayesian Information Criterion

1.5.4 Takeuchi Information Criterion

1.6 Decision Theory

The joint distribution $p(\mathbf{X}, \mathbf{Y})$ completely summarizes of the uncertainty associated with these variables, and determination of $p(\mathbf{X}, \mathbf{Y})$ from a training dataset is called **inference**, which is generally a very hard problem. In practical application, we must often make a specific prediction for the value of $\mathbf{y}$, or more generally take a specific action based on the value of $\mathbf{y}$, and this subject is called decision theory.

Definition 1.1 (Loss Function)**.** Let us construct a **loss function** $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ that defines the loss we incur if we use $h_{\mathbf{w}}(\mathbf{x})$ to predict the true value $\mathbf{y}$.

Example 1.4. Here are some popular examples of loss functions.

1. In regression, we can take the squared loss

$$L\big(\mathbf{y}, h_{\mathbf{w}}(\mathbf{x})\big) \coloneqq \big(\mathbf{y} - h_{\mathbf{w}}(\mathbf{x})\big)^2$$

2. Let us look at a classification problem in which Y can take K values, known as classes. That is, we can let Y be a discrete random variable taking values in $k = 1, \dots, K$. We would like a rule that divides region $\mathcal{X}$ into decision regions $\mathcal{R}_k$, so that if $\hat{\mathbf{x}} \in \mathcal{R}_k$, then we will predict that $\hat{\mathbf{y}} = k$. Now if we want

to minimize the misclassifications, or equivalently, maximize the probability of classifying correctly, then we want the loss function to simply model the number of incorrect guesses and we define

$$L\big(\mathbf{y}, h_{\mathbf{w}}(\mathbf{x})\big) \coloneqq \begin{cases} 0 & \text{if } \mathbf{y} = h_{\mathbf{w}}(\mathbf{x}) \\ 1 & \text{if } \mathbf{y} \neq h_{\mathbf{w}}(\mathbf{x}) \end{cases}$$

We can organize this into a loss matrix for notational convenience.

3. Sometimes, misclassifying $h_{\mathbf{w}}(\mathbf{x}) = k$ when $\mathbf{y} = k'$ may be worse than misclassifying $h_{\mathbf{w}}(\mathbf{x}) = k'$ when $\mathbf{y} = k$. Therefore, we may have to adjust the loss function above to

$$L\big(\mathbf{y}, h_{\mathbf{w}}(\mathbf{x})\big) \coloneqq \begin{cases} 0 & \text{if } \mathbf{y} = h_{\mathbf{w}}(\mathbf{x}) \\ K_{ij} & \text{if } \mathbf{y} = i \neq j = h_{\mathbf{w}}(\mathbf{x}) \end{cases}$$

Therefore, given that we have a hypothesis function, we can set $\mathbf{x}, \mathbf{y}$ to the random variables X, Y. If we have completed the task of inference, i.e. have computed an approximation of $p(X, Y)$, then we can simply take the expectation

$$\mathbb{E}_{X\times Y}[L] = \mathbb{E}[L(Y, h_{\mathbf{w}}(X))] = \int\int L\big(\mathbf{y}, h_{\mathbf{w}}(\mathbf{x})\big)\, p(x, Y = y)\, d\mathbf{x}\, d\mathbf{y}$$

to compute the expected loss. In a classification problem, this would reduce to

$$\begin{aligned} \sum_k \int L\big(\mathbf{y}, h_{\mathbf{w}}(\mathbf{x})\big)\, p(x, Y = y)\, d\mathbf{x} &= \sum_k \sum_j L(k, j) \cdot p(\mathbf{x} \in \mathcal{R}_k, Y = j) \\ &= \sum_k \sum_j L(k, j) \int_{\mathcal{R}_k} p(\mathbf{x}, Y = j)\, d\mathbf{x} \end{aligned}$$

Now if we just wanted to minimize the misclassification rate, then we would have $L(k, j) = -1$ if $k \neq j$ and 0 if $k = j$, and the above would reduce to

$$\sum_{k,j;k\neq j} \int_{\mathcal{R}_k} p(\mathbf{x}, Y = j)\, dx = \sum_{k,j;k\neq j} \int_{\mathcal{R}_k} p(Y = j \mid \mathbf{x})\, p(\mathbf{x})\, dx$$

but by complementation, it is slightly easier to maximize the analogous probability of being correctly classified, which can be further divided by the common term $p(\mathbf{x})$

$$\sum_{k=1}^K \int_{\mathcal{R}_k} p(\mathbf{x}, Y = k)\, d\mathbf{x} = \sum_{k=1}^K \int_{\mathcal{R}_k} p(Y = k \mid \mathbf{x})\, p(\mathbf{x})\, d\mathbf{x} = \left(\sum_{k=1}^K \int_{\mathcal{R}_k} p(Y = k \mid \mathbf{x})\, d\mathbf{x} \right) p(\mathbf{x})$$

and to maximize this, we should assign $\mathbf{x}$ to be in the class having the largest posterior probability $p(Y = k \mid \mathbf{x})$.

Now let's go to the regression case. Given the squared loss function L, we basically want to minimize $\mathbb{E}[L]$ over the joint measure on $X \times Y$. To do this, let us decompose the expectation as such:

$$\begin{aligned} \mathbb{E}[L] &= \mathbb{E}\big[\big(Y - g(X)\big)^2\big] \\ &= \mathbb{E}\big[\big((Y - \mathbb{E}[Y \mid X]) + (\mathbb{E}[Y \mid X] - g(X))\big)^2\big] \\ &= \mathbb{E}[\big(Y - \mathbb{E}[Y \mid X]\big)^2] + \mathbb{E}[(Y - \mathbb{E}[Y \mid X])\mathbb{E}[Y \mid X] - g(X)] + \mathbb{E}[\big(\mathbb{E}[Y \mid X] - g(X)\big)^2] \\ &= \mathbb{E}[\big(Y - \mathbb{E}[Y \mid X]\big)^2] + \mathbb{E}[\big(\mathbb{E}[Y \mid X] - g(X)\big)^2] \end{aligned}$$

where the middle term cancels out due to the tower property

$$TBD$$

Therefore, we know that the squared loss of g basically decomposes into the squared loss of $\mathbb{E}[Y \mid X]$ and $g(X)$, which is the intrinsic misspecification of the model, plus the squared difference of Y with its best approximation $\mathbb{E}[Y \mid X]$, which is the intrinsic noise inherent in Y beyond the σ-algebra of X.

We also proved a second fact: Since $\mathbb{E}[(Y - \mathbb{E}[Y \mid X])^2]$ is the misspecification of the model, we cannot change this (positive) constant, so $\mathbb{E}\big[\big(Y - g(X)\big)^2\big] \geq \mathbb{E}[(Y - \mathbb{E}[Y \mid X])^2]$, with equality achieved when we perfectly fit g as $\mathbb{E}[Y \mid X]$ (i.e. the model is well-specified). Therefore, denoting $\mathcal{F}$ as the set of all $\sigma(X)$-measurable functions, then the minimum of the loss is attained when

$$\operatorname*{argmin}_{g \in \mathcal{F}} \mathbb{E}[L] = \operatorname*{argmin}_{g \in \mathcal{F}} \mathbb{E}\big[\big(Y - g(X)\big)^2\big] = \mathbb{E}[Y \mid X]$$

1.7 Information Theory

The **entropy** of a *discrete* random variable X is defined

$$H[X] \coloneqq \mathbb{E}[-\ln p(X)] = -\sum_x p(x) \ln p(x)$$

the base of the logarithm can be any number, but we will define it to be the natural logarithm. Intuitively, this represents the element of surprise of a certain data point, and distributions that have relatively sharp peaks will have lower entropy (since we expect most of the samples to come from the peaks) while uniform distributions have higher entropy. The entropy also demonstrates the average length (if base is 2) number of bits required to transmit the state of a random variable.

For a continuous random vector, the **differential entropy** is defined

$$H[\mathbf{X}] = -\int p(\mathbf{x}) \ln p(\mathbf{x})\, d\mathbf{x}$$

Now, the **conditional entropy** of $\mathbf{Y}$ given $\mathbf{X}$ is defined

$$H[\mathbf{Y} \mid \mathbf{X}] \coloneqq -\int \int p(\mathbf{y}, \mathbf{x})\ \ln p(\mathbf{y} \mid \mathbf{x})\, d\mathbf{y} d\mathbf{x}$$

1.7.1 Relative Entropy and Mutual Information

The **relative entropy**, or **Kullback-Leibler divergence**, of distributions $p(x)$ and $q(x)$ is defined

$$\begin{aligned} \mathrm{KL}(p||q) &\coloneqq -\int p(\mathbf{x})\ \ln q(\mathbf{x})\, d\mathbf{x} - \left(-\int p(\mathbf{x})\ \ln p(\mathbf{x})\, d\mathbf{x} \right) \\ &= -\int p(\mathbf{x})\ \ln \left(\frac{q(\mathbf{x})}{p(\mathbf{x})} \right) d\mathbf{x} \end{aligned}$$

We can show that this quantity is always greater than or equal 0 by Jensen's inequality using the fact that $-\ln(x)$ is concave

$$\int p(\mathbf{x})\ -\ln \left(\frac{q(\mathbf{x})}{p(\mathbf{x})} \right) d\mathbf{x} \geq -\ln \int p(\mathbf{x})\, \frac{q(\mathbf{x})}{p(\mathbf{x})}\, d\mathbf{x} = -\ln \int q(\mathbf{x})\, d\mathbf{x} = -\ln(1) = 0$$

and it is precisely 0 if $p = q$, so it behaves similarly to a metric. However, it isn't exactly since it is not symmetric.

1.8 Standardizing Data

Standardizing typically meanss that our featuers willbe rescaled to have the properties of a standard normal distribution with mean of 0 and a standard deviation of 1. Here are a few methods to scale our data, with their results shown on a dataset of 30 points in $\mathbb{R}^2$.

1. **StandardScaler**: This is probably the most used method for standardizing data. It standardizes features by removing the mean and scaling to unit variance. The standard score of a sample $x^{(n)}$ is $(x - \bar{x})/S$ where $\bar{x}$ is the mean of the training samples and S is the standard deviation of the training samples.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

2. **MinMaxScaler**: While not technically "standardization," MinMaxScaler is another preprocessing method for scaling. It transforms features by scaling each feature to a given range, typically between zero and one, or so that the maximum absolute value of each feature is scaled to unit size.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
```

3. **MaxAbsScaler**: This scaler works similarly to the MinMaxScaler but scales in a way that the training data lies within the range $[-1, 1]$ by dividing through the largest maximum value in absolute value. It is meant for data that is already centered at zero or sparse data.

```
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
scaled_data = scaler.fit_transform(data)
```

4. **RobustScaler**: This scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). It's robust to outliers, which makes it a good choice if you have data with possible outliers.

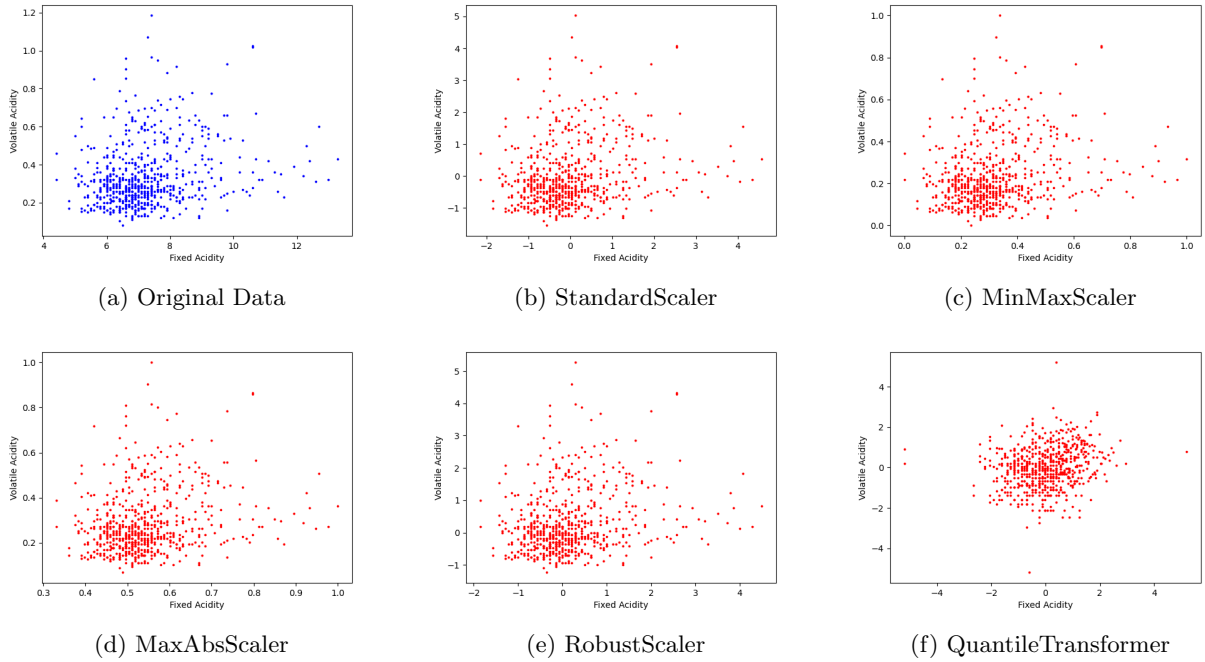
```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
scaled_data = scaler.fit_transform(data)
```

5. **QuantileTransformer**: Note that the presence of outliers messes with our scaling. More generally for skewed distributions (like an exponential), a linear transformation does not take care of these outliers, so we would like some nonlinear preprocessing algorithm. One common one is the QuantileTransformer, which takes the quantiles (percentiles) of the dataset and transforms then so that those are equidistant from each other. By default, it divides up the data into 1000 quantiles.

```
from sklearn.preprocessing import QuantileTransformer
transformer = QuantileTransformer(n_quantiles = 100, output_distribution='normal')
transformed_data = transformer.fit_transform(data)
```

Let's talk about how these scalers will work on some data. We take a wine data with the two variables representing fixed acidity and volatile acidity.

It's important to note that whether you should standardize your data and how you should do it depends on the specific characteristics of your data and the machine learning algorithm you're using. For example, some algorithms, like many in deep learning, assume that all features are on the same scale. Others, like Decision Trees and Random Forests, do not require feature scaling at all.



(a) Original Data (b) StandardScaler (c) MinMaxScaler

(d) MaxAbsScaler (e) RobustScaler (f) QuantileTransformer

Figure 7: The StandardScaler simply standardizes the data to have 0 mean and unit variance.

1.9 Data Imputation

Data imputation refers to filling out missing values in our data or our design matrix. For now, we will refer to the most simple data imputation method, which takes the mean of the rest of the available data within the column.

```
from sklearn.impute import SimpleImputer
X = np.array([[7, 2, 3], [4, np.nan, 6], [10 ,5, 9]])

imp = SimpleImputer(strategy="mean")
transformed_data = imp.fit_transform(data)

print(transformed_data)
# [[ 7.  2.  3.]
#  [ 4.  3.5 6.]
#  [10.  5.  9.]]
```

Therefore, the general workflow of any statistical modeling is:

1. Preprocess the data: data imputation, standardization, and feature transformation on the covariates $\mathbf{x}^{(i)}$, and perhaps encode the $\mathbf{y}^{(i)}$ if they are categorical.
2. Choose a class of models $\mathcal{M}_1, \dots, \mathcal{M}_K$
3. Train and fit the models
4. Choose the model that is best by tuning hyperparameters, using regularization, and performing K-fold cross validation

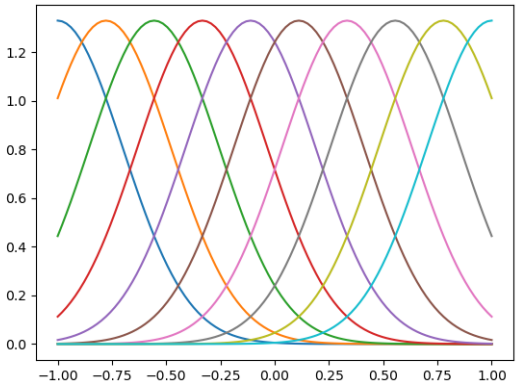


Figure 8: Gaussian basis functions over the interval $[-1, 1]$ with standard deviation of 0.3

2 Linear Models for Regression

2.1 Linear Basis Function Models

The simplest linear function for regression is simply

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

This is called linear regression not because h is a linear function of $\mathbf{x}$. It is a linear function of $\mathbf{w}$. Therefore, we can fix nonlinear functions $\phi_j(\mathbf{x})$ and consider linear combinations of them.

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

We usually choose a dummy basis function $\phi_0(\mathbf{x}) = 1$ for notational convenience, so that if $\boldsymbol{\phi}$ is the vector of the function ϕ_j, then we can write $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$. This mapping from the original variables $\mathbf{x} \in \mathbb{R}^D$ to the basis functions $\{\phi_j(\mathbf{x})\}$, which span a linear function space of dimension M, is called **preprocessing** or **feature extraction** of the data.

Example 2.1. Here are some examples of how we can extract features.

1. The mapping from a single variable x to its powers

$$x \mapsto (1, x, x^2, \ldots, x^{M-1})$$

2. The mapping from a configuration of K atoms with their momenta in $\mathbb{R}^{6K}$ to their atomic cluster expansion polynomials.

3. The legendre polynomials, which form an orthonormal basis in the space of polynomials.

4. Using equally spaced Gaussian basis functions over the dataset as shown in Figure 8.

Changing the input space from D dimensions to M dimensions (i.e. extracting our M features) gives the design matrix

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \mathbf{x}^{(3)} \\ \vdots \\ \mathbf{x}^{(n)} \end{pmatrix} \implies \boldsymbol{\Phi} = \begin{pmatrix} — & \phi(\mathbf{x}^{(1)}) & — \\ — & \phi(\mathbf{x}^{(2)}) & — \\ \vdots & \vdots & \vdots \\ — & \phi(\mathbf{x}^{(n)}) & — \end{pmatrix}$$

Now assuming that the $\mathbf{Y}$ is given by a deterministic function $h_{\mathbf{w}}$ with Gaussian noise $\epsilon \sim N(0, \sigma^2)$, we can write

$$p(y \mid \mathbf{x}, \mathbf{w}, \sigma^2) = N(y \mid h_{\mathbf{w}}(\mathbf{x}), \sigma^2)$$

and now considering the distribution of all $\mathbf{Y}$ given $\mathbf{X}$, which by independence is the product, we get

$$p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{n=1}^{N} N(y^{(n)} \mid \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}^{(n)}), \sigma^2)$$

and taking the logarithm

$$\ln p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w}, \sigma^2) = -\frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} \left(y^{(n)} - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}^{(n)})\right)^2$$

which then corresponds to minimizing the sum of squares error function $E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(y^{(n)} - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}^{(n)})\right)^2$. Taking the gradient of this log likelihood w.r.t. $\mathbf{w}$ gives

$$0 = \sum_{n=1}^{N} y^{(n)} \boldsymbol{\phi}(\mathbf{x}^{(n)})^T - \mathbf{w}^T \left(\sum_{n=1}^{N} \boldsymbol{\phi}(\mathbf{x}^{(n)}) \boldsymbol{\phi}(\mathbf{x}^{(n)})^T \right)$$

which is equivalent to solving the equation

$$\mathbf{w}_{ML} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{Y}$$

Note that if we write out the bias term out explicitly, we can see that it just accounts for the translation (difference) between the average of the outputs $\bar{y} = \frac{1}{N} \sum_{n=1}^{N} y_n$ and the average of the basis functions $\bar{\phi_j} = \frac{1}{N} \sum_{n=1}^{N} \phi_j(\mathbf{x}^{(n)})$.

$$w_0 = \bar{y} - \sum_{j=1}^{M-1} w_j \bar{\phi_j}$$

We can also maximize the log likelihood w.r.t. σ^2, which gives the MLE

$$\sigma^2_{ML} = \frac{1}{N} \sum_{n=1}^{N} \left(y^{(n)} - \mathbf{w}_{ML}^T \boldsymbol{\phi}(\mathbf{x}^{(n)})\right)^2$$

Another way to solve this is to use an iterative method like gradient descent, which computes the gradient of the cost function (which points in the direction of steepest ascent) and goes in the opposite direction. Therefore, we can initialize a $\mathbf{w}^{(0)}$ anywhere and iteratively solve

$$\mathbf{w}^{(j+1)} = \mathbf{w}^{(j)} - \alpha \nabla E(\mathbf{w})$$

where α is the **learning rate**, which can be monotonically decreased to encourage convergence. For a general function $h_{\mathbf{w}}$, this requires us to compute

$$TBD$$

but if h is linear w.r.t. $\mathbf{w}$, then this gradient simplifies to

$$TBD$$

Moreover, the linearity of h guarantees that $E_D(\mathbf{w})$ is convex (proof?), which means that we do not have to worry about the GD getting stuck at local minima.

Usually solving the normal equations is not an efficient way since we have to iterate through the entire batch of N inputs. With batch gradient descent, we have the compute the gradient as the sum over the entire batch

$$TBD$$

which may also not be efficient, so instead we take random minibatch of the N samples and compute the gradient for that batch. This demonstrates that taking many unoptimal steps is better than taking slow optimal ones.

2.1.1 Transformers in Scikit-Learn

We have shown that the `PolynomialFeatures` transformer converts our features to a polynomial basis. We can do this for an arbitrary number of features, for example if we map $D = 2$ to a second degree polynomial, we would have the transformation

$$(x_1, x_2) \mapsto (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$$

```
>>> import numpy as np
>>> from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

Sometimes, we are only worried about the interaction terms among features, so we can set the parameter `interaction_only=True`, which would, in the third degree case, transform the features

$$(x_1, x_2, x_3) \mapsto (1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1x_2x_3)$$

Spline transformers are piecewise polynomials, which is also built in. We notice that it is cumbersome to transform the dataset `X` with the transformer, store it into another variable, and train the model on that. We can "combine" the transforming (even multiple layers of transformers) and the model by implementing a "pipeline," which is initialized by inputting a list of tuples (name and the object) and has the same methods as the model.

```
from sklearn.pipeline import Pipeline
model = Pipeline([("poly_transform", PolynomialFeatures(degree=2)),
                  ("lin_regression", LinearRegression())
                  ])
model.fit(X, y)
```

Now, let's talk about how we can implement a custom transformer. We basically have to create a new subclass that implements the `fit` (which always returns `self`) and the `transform` (which returns the transformed matrix) methods. Here we show for Gaussian basis functions.

```
from sklearn.base import BaseEstimator, TransformerMixin

class GaussianFeatures(BaseEstimator, TransformerMixin):
    """Uniformly spaced Gaussian features for one-dimensional input"""

    def __init__(self, N, width_factor=2.0):
        self.N = N
        self.width_factor = width_factor

    def fit(self, X, y=None):
        # create N centers spread along the data range
        self.centers_ = np.linspace(X.min(), X.max(), self.N)
        self.width_ = self.width_factor * (self.centers_[1] - self.centers_[0])
        return self

    def transform(self, X):
        transformed_rows = []
        for mu in self.centers_:
            transformed_rows.append(stats.norm.pdf(X, mu, self.width_))

        return np.hstack(tuple(transformed_rows))

model = Pipeline([("gauss_transform", GaussianFeatures(20)),
                  ("lin_regression", LinearRegression())
                  ])

N = 60
X = np.random.uniform(-1, 1, size=(N, 1))
Y = true_func(X) + np.random.normal(0, 0.3, size=(N, 1))

model = Pipeline([("gauss_transform", GaussianFeatures(10)),
                ("lin_regression", LinearRegression())
                ])
model.fit(X, Y)
```

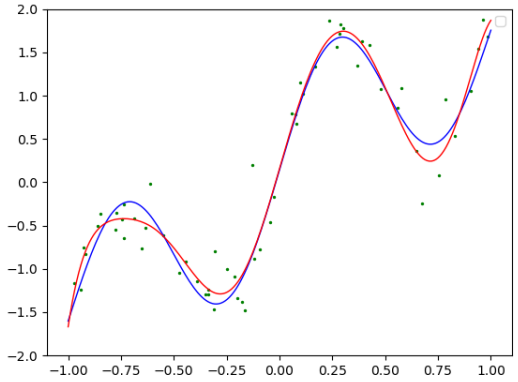
If we would like to impelment the fourier expansion of a function of form

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{N} a_n \cos(nx) + \sum_{n=1}^{N} b_n \sin(nx)$$

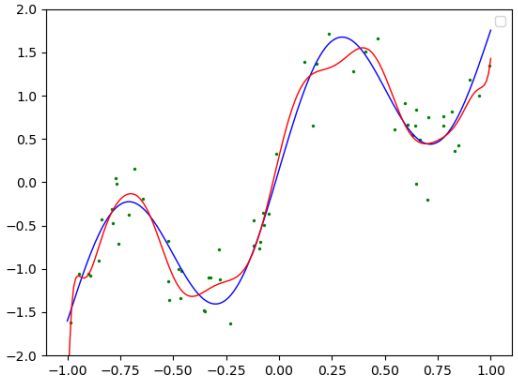
Then we would create the basis functions according to

```
class FourierFeatures(BaseEstimator, TransformerMixin):
    "Fourier Expansion for one-dimensional input"

    def __init__(self, N):
        self.N = N

    def fit(self, X, Y=None):
        return self

    def transform(self, X):
        transformed_columns = []
        transformed_columns.append(np.ones(shape=X.shape))

        for n in range(self.N):
            transformed_columns.append(np.sin(n * X))
            transformed_columns.append(np.cos(n * X))

        print(np.hstack(tuple(transformed_columns)).shape)
        return np.hstack(tuple(transformed_columns))
```

and both of them would give the following fits to our original function $f(x) = \sin(2\pi x) + 2\cos(x - 1.5)$.



(a) Fitting with 10 Gaussian basis functions.



(b) Fitting with 10 Fourier basis functions.

Figure 9

2.2 Regularization

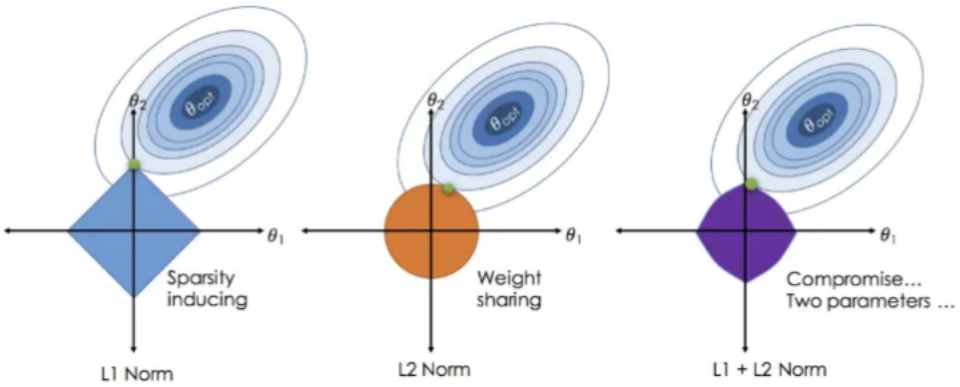
Now we can also add a regularization term in order to control overfitting, of the form $E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$. E_W is a monotonically increasing function of some norm of $\mathbf{w}$ and so if $||\mathbf{w}||_q$ is big, then it penalizes the model more. Here we discuss 3 types of regularizers and the difference between them.

1. The ridge regularizer $E_W(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||_2^2$ can be imagined as drawing equipotential circles in $\mathcal{W}$.

2. The LASSO regularizer $E_W(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||_1 = \frac{1}{2}\sum_{j=1}^{M} |w_j|$ draws a diamond in the parameter space $\mathcal{W}$. This tends to give a sparser solution since the error function E_D is more likely to "touch" the corners of the contour plots of E_W.

3. The elastic net is a linear combination of the ridge and LASSO: $E_W(\mathbf{w}) = \lambda_2||\mathbf{w}||_2^2 + \lambda_1||\mathbf{w}||_1$.

Therefore, we can effectively limit the model complexity by adding a regularization to our error function so we don't have to worry about the number of basis functions M to choose, but this now raises the question of how to determine a suitable regularization parameter λ.



2.2.1 Regularizers in Scikit-Learn

We can perform ridge, lasso, and elastic net regression very easily in sklearn, as such:

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet
model1 = Ridge(alpha=0.1).fit(X, Y)
model2 = Lasso(alpha=0.01).fit(X, Y)

# l1_ratio determines the balance between L1 and L2 regression
model3 = ElasticNet(alpha=0.5, l1_ratio=0.5)
```

2.3 Bias Variance Decomposition

Recall that the expectation of the squared loss for a regression function can be decomposed as

$$\mathbb{E}_{X\times Y}\big[\big(Y - h_{\mathbf{w}}(X)\big)^2\big] = \mathbb{E}_{X\times Y}[\big(Y - \mathbb{E}[Y \mid X]\big)^2] + \mathbb{E}_X[\big(\mathbb{E}[Y \mid X] - h_{\mathbf{w}}(X)\big)^2]$$

Now if we approximate $\mathbb{E}[Y \mid X]$ with our parameterized hypothesis $h_{\mathbf{w}}$, then from a Bayesian perspective the uncertainty in our model is expressed through a poseterior distribution over $\mathbf{w}$.

A frequentist treatment, however, involves making a point estimate of $\mathbf{w}$ based on the dataset $\mathcal{D}$ and tries instead to interpret the uncertainty of this estimate through the following thought experiment: Suppose we had a large number of datasets each of size N and each drawn independently from the joint distribution $X \times Y$. For any given dataset $\mathcal{D}$, we can run our learning algorithm and obtain our best fit function $h^*_{\mathbf{w};\mathcal{D}}(\mathbf{x})$. Different datasets from the ensemble will give different functions and consequently different values of the squared loss. The performance of a particular learning algorithm is then assessed by taking the average over this ensemble of datasets, which we define $\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})] = \mathbb{E}_{(X\times Y)^N}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})]$. We are really taking an expectation over all datasets, meaning that the N points in each $\mathcal{D}$ must be sampled from $(X \times Y)^N$.

Consider the term $\big(\mathbb{E}[Y \mid X] - h_{\mathbf{w}}(X)\big)^2$ above, which models the discrepancy in our optimized hypothesis and the best approximation. Now, over all datasets $\mathcal{D}$, there will be a function $h_{\mathbf{w};\mathcal{D}}$, and averaged over all datasets $\mathcal{D}$ is $\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}]$. So, the random variable below (of $\mathcal{D}$ and X) representing the stochastic difference between our optimized function $h_{\mathbf{w};\mathcal{D}}(X)$ and our best approximation $\mathbb{E}[Y \mid X]$ can be decomposed into

$$\begin{aligned}
\big(\mathbb{E}[Y \mid X] - h_{\mathbf{w}:\mathcal{D}}(X)\big)^2 &= \big[\big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})]\big) + \big(\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})] - h_{\mathbf{w}:\mathcal{D}}(X)\big)\big]^2 \\
&= \big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})]\big)^2 + \big(\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})] - h_{\mathbf{w}:\mathcal{D}}(X)\big)^2 \\
&\quad + 2\big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})]\big)\big(\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})] - h_{\mathbf{w}:\mathcal{D}}(X)\big) \\
&= \big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})]\big)^2 + \big(\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})] - h_{\mathbf{w}:\mathcal{D}}(X)\big)^2
\end{aligned}$$

Averaging over all datasets $\mathcal{D}$ causes the middle term to vanish and gives us the expected squared difference between the two random variables, now of X.

$$TBD$$

Therefore, we can write out the expected square difference between $h_{\mathbf{w}}$ and $\mathbb{E}[Y \mid X]$ as the sum of two terms.

$$\mathbb{E}_{\mathcal{D}}\big[\big(\mathbb{E}[Y \mid X] - h_{\mathbf{w}}(X)\big)^2\big] = \underbrace{\big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(X)]\big)^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\big[\big(\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(X)] - h_{\mathbf{w}}(X)\big)^2\big]}_{\text{variance}}$$

Let us observe what these terms mean:

1. The **bias** $\mathbb{E}[Y \mid X = \mathbf{x}] - \mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})]$ measures the difference in how the average prediction of our hypothesis function $\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})]$ differs from the actual prediction $\mathbb{E}[Y \mid X = \mathbf{x}]$.

2. The **variance** $\mathbb{E}_{\mathcal{D}}\big[\big(\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})] - h_{\mathbf{w};\mathcal{D}}(X)\big)^2\big]$ measures the variability of each hypothesis function $h_{\mathbf{w}}(X)$ about its mean over the ensemble $\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(\mathbf{x})]$.

Therefore, by the law of iterated expectations, we can substitute this back into the expected squared loss to get

$$\text{Expected Loss} = (\text{Bias})^2 + \text{Variance} + \text{Noise}$$

where

$$\begin{aligned}(\text{Bias})^2 &= \mathbb{E}_X\big[\big(\mathbb{E}[Y \mid X] - \mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(X)]\big)^2\big] \\ \text{Variance} &= \mathbb{E}_X\big[\mathbb{E}_{\mathcal{D}}\big[\big(\mathbb{E}_{\mathcal{D}}[h_{\mathbf{w};\mathcal{D}}(X)] - h_{\mathbf{w}}(X)\big)^2\big]\big] \\ \text{Noise} &= \mathbb{E}_{X\times Y}[\big(Y - \mathbb{E}[Y \mid X]\big)^2]\end{aligned}$$

Ultimately, this is theoretical explanation for underfitting and overfitting of data. In practice, we don't have an emsemble of datasets; we just have one. Therefore, we don't actually know what the bias, the variance, or the noise is at all. All we can do is adjust the model complexity or the regularization parameter and observe how this affects the error on the testing set (since the error on the training set will always go down with higher complexity), and once it starts going up, that's when we should stop.

3 Linear Models for Classification

3.1 Discriminative Models

Recall that a discriminant function is a function that directly maps an input vector $\mathbf{x}$ to a class $\mathcal{C}_k$. We have used numbers $k = 1, 2, \dots, K$ to represent classes, but assigning a numerical value on the classes is not desirable. Therefore, we can use **one-hot-encoding**, which refers to mapping the class to the standard unit vectors with 1 in the kth element and 0 everywhere else.

$$\mathcal{C}_k \mapsto e_k$$

3.1.1 Perceptron Algorithm

Now to do binary classification, we can also use the **perceptron algorithm**. That is, given our extracted features $\boldsymbol{\phi}(\mathbf{x}^{(n)})$ with $\boldsymbol{\phi}(\mathbf{x}^{(n)})_0 = 1$ for the bias variable, we can model our hypothesis function as

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) \geq 0 \\ -1 & \text{if } \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) < 0 \end{cases}$$

where we have chosen to label class $\mathcal{C}_1 = 1$ and $\mathcal{C}_2 = -1$. We can optimize our parameter $\mathbf{w}$ by minimizing a cost function, which we may be tempted to write as simply the number of misclassifications

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} |y^{(n)} - h_{\mathbf{w}}(\boldsymbol{\phi}^{(n)})|$$

but the gradient of this function is zero almost everywhere, so it is not useful to optimize. Rather, we would want a function that penalizes not only if there is a misclassification, but how *far* that misclassified point is from the boundary. Therefore, if $y^{(n)}$ and $h_{\mathbf{w}}(\mathbf{x}^{(n)})$ have the same sign, i.e. if $y^{(n)}h_{\mathbf{w}}(\boldsymbol{\phi}^{(n)}) > 0$, then the penalty should be 0, and if it is < 0, then the penalty should be proportional to the orthogonal distance of the misclassified point to the boundary, which is represented by $-\mathbf{w}^T\boldsymbol{\phi}^{(n)}y^{(n)}$ (where the negative sign makes this cost term positive). To define our final cost function, let us define the function

$$[f(\mathbf{x})]_+ \coloneqq \begin{cases} f(\mathbf{x}) & \text{if } f(\mathbf{x}) > 0 \\ 0 & \text{else} \end{cases}$$

Therefore, our cost functions would take all the points and penalize all the terms by 0 if they are correctly classified and by $-\mathbf{w}^T\boldsymbol{\phi}^{(n)}y^{(n)}$ if incorrectly classified.

$$E(\mathbf{w}) = \sum_{n=1}[-\mathbf{w}^T\boldsymbol{\phi}^{(n)}y^{(n)}]_+$$

which is a piecewise linear function and therefore can be optimized. In fact, it can be proven that if a dataset is **linearly separable**, i.e. if there exists an affine hyperplane separating the two classes, then the perceptron is guaranteed to find it. Therefore, the gradient descent algorithm

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w})$$

which is easier to interpret as SGD updating one partial at a time

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\frac{\partial E(\mathbf{w})}{\partial w_n}\bigg|_{\mathbf{w}^{(\tau)}}$$

where by abuse of notation, the partial represents the vector of 0's except for the nth term, will convergence to a solution if one exists. However, it does not guarantee that every step of GD will reduce the error.

Let's implement this in scikit-learn, using two pipelines with different data standardization techniques to see the differences in the perceptron boundary.

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import QuantileTransformer, StandardScaler

pipe1 = Pipeline([
    ("scale", StandardScaler()),
    ("model", Perceptron())
])

pipe2 = Pipeline([
    ("scale", QuantileTransformer(n_quantiles=100)),
    ("model", Perceptron())
])
```

which results in the following shown in Figure 10.

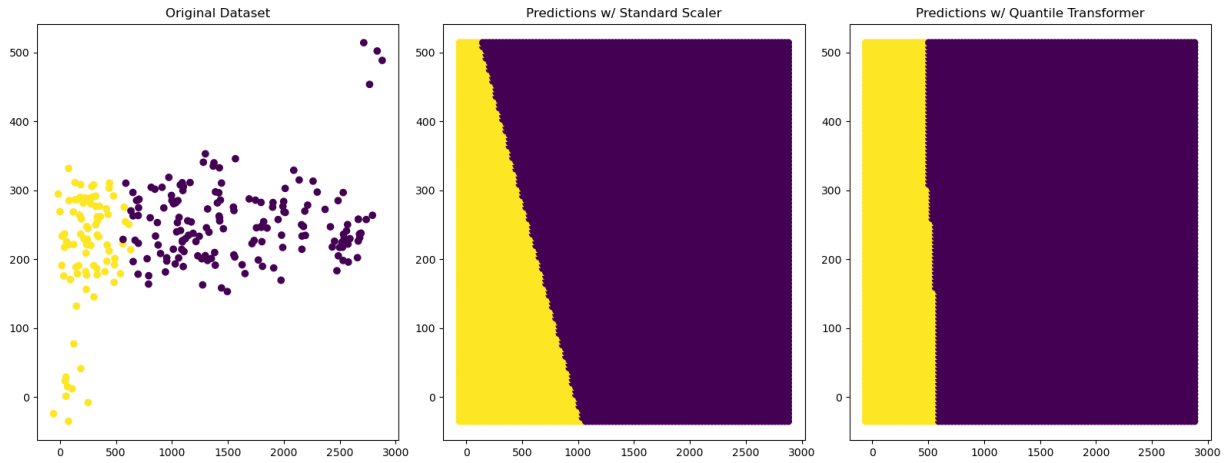


Figure 10: Perceptron Trained on Different Standardized Data

3.2 Generative Models

3.2.1 Sigmoid and Softmax Functions

3.2.2 Gaussian Discriminant Analysis

GDA assumes that $\mathbb{P}(x\,|y)$ is distributed according to a multivariate Gaussian distribution. Let us assume that the input space is d-dimensional and this is a binary classification problem. We set

$$\begin{aligned} y &\sim \text{Bernoulli}(\pi) \\ x\,|\,y=0 &\sim \mathcal{N}_d(\mu_0, \Sigma) \\ x\,|\,y=1 &\sim \mathcal{N}_d(\mu_1, \Sigma) \end{aligned}$$

This method is usually applied using only one covariance matrix Σ. The distributions are

$$\begin{aligned} p(y) &= \pi^y(1-\pi)^{1-y} \\ p(x\,|\,y=0) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right) \\ p(x\,|\,y=1) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right) \end{aligned}$$

Now, what we have to do is optimize the distribution parameters $\pi \in (0,1)\mathbb{R}, \mu_0 \in \mathbb{R}^d, \mu_1 \in \mathbb{R}^d, \Sigma \in \text{Mat}(d\times d, \mathbb{R}) \simeq \mathbb{R}^{d\times d}$ so that we get the best-fit model. Assuming that each sample has been picked independently, this is equal to maximizing

$$L(\pi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^{n} \mathbb{P}\big(x^{(i)}, y^{(i)}\,;\,\pi, \mu_0, \mu_1, \Sigma\big)$$

which is really just the probability that we get precisely all these training samples $(x^{(i)}, y^{(i)})$ given the 4 parameters. This can be done by optimizing its log-likelihood, which is given by

$$\begin{aligned} l(\pi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} \mathbb{P}(x^{(i)}, y^{(i)}; \pi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} \mathbb{P}(x^{(i)}\,|\,y^{(i)}; \mu_0, \mu_1, \Sigma)\,\mathbb{P}(y^{(i)}; \pi) \\ &= \sum_{i=1}^{n} \log\left(\mathbb{P}(x^{(i)}\,|\,y^{(i)}; \mu_0, \mu_1, \Sigma)\,\mathbb{P}(y^{(i)}; \pi)\right) \end{aligned}$$

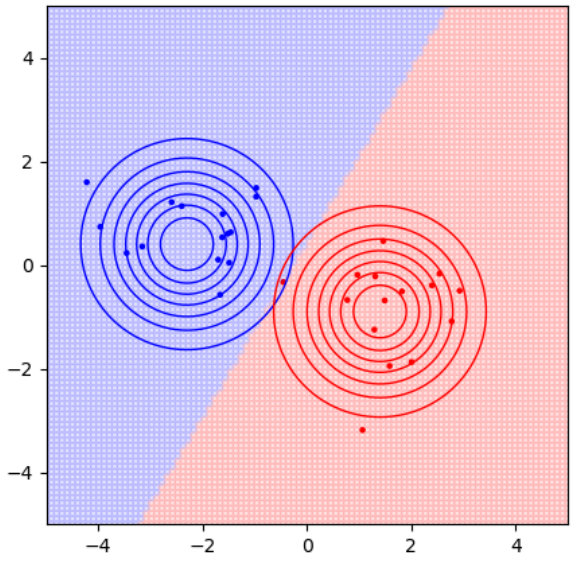


Figure 11: GDA of Data Generated from 2 Gaussisans centered at $(-2.3, 0.4)$ and $(1.4, -0.9)$ with unit covariance. The decision boundary is slightly off since MLE approximates the true means.

and therefore gives the maximum likelihood estimate to be

$$\begin{aligned}
\pi &= \frac{1}{N}\sum_{n=1}^{N} 1\{y^{(n)} = 1\} \\
\boldsymbol{\mu}_0 &= \frac{\sum_{n=1}^{n} 1_{\{y^{(n)}=0\}}\mathbf{x}^{(n)}}{\sum_{n=1}^{N} 1_{\{y^{(n)}=0\}}} \\
\boldsymbol{\mu}_1 &= \frac{\sum_{n=1}^{n} 1_{\{y^{(n)}=1\}}\mathbf{x}^{(n)}}{\sum_{n=1}^{N} 1_{\{y^{(n)}=1\}}} \\
\boldsymbol{\Sigma} &= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}^{(n)} - \mu_{y^{(n)}})(\mathbf{x}^{(n)} - \mu_{Y^{(i)}})^T
\end{aligned}$$

A visual of the algorithm is below, with contours of the two Gaussian distributions, along with the straight line giving the decision boundary at which $\mathbb{P}(y = 1 \,|\, x) = 0.5$.

3.2.3 Logistic Regression

The perceptron algorithm is a function that takes an input value and outputs either 0 or 1, but there is no sense of uncertainty¡. What if we wanted a probability of an input vector being in class 0 or 1? Rather than outputting either 0 or 1, an algorithm that outputs a 2-tuple $(p, 1-p)$ that represents the probability that it is in 0 and 1 (e.g. $(0.45, 0.55)$). This is where **logistic regression** comes in. Recall that the formula for the logistic function $f : \mathbb{R} \longrightarrow (0, 1)$ is:

$$f(x) \equiv \frac{L}{1 + e^{-k(x-x_0)}}$$

where L is the curve's maximum value, k is the logistic growth rate or steepness of the curve, and x_0 is the value of the sigmoid midpoint. What makes the logistic function right for this type of problem is that it's

range is $(0, 1)$, with the asymptotic behavior towards $y = 0$ and $y = 1$. We can adjust our hypothesis h to the logistic equation

$$h_\theta(x) \equiv g\big(\theta^T x\big) \equiv \frac{1}{1 + e^{-\theta^T x}}$$

Note that this is just a composition of the functions

$$x \mapsto \theta^T x \quad \text{and} \quad x \mapsto \frac{1}{1 + e^{-x}}$$

There is a very nice way to visualize this. Consider the one-dimensional logistic function

$$g(z) \equiv \frac{1}{1 + e^{-z}}$$

which has range $(0, 1)$ and $g(z) \to 1$ and $z \to \infty$ and $g(z) \to 0$ as $z \to -\infty$. Rather than imagining it as the regular sigmoid in $\mathbb{R}^2$, we can just imagine the domain $\mathbb{R}$ itself and "color" it in greyscale, representing 0 as white and 1 as black. Then, we should visualize the real line as a gradient that gets more and more white (but not completely white) as $z \to -\infty$ and more and more black (but not completely black) as $z \to \infty$. We extend this into a d-dimensional space. Given the d-dimensional point x (note that we did ¡i¿not¡/i¿ set $x_0 = 0$), the function (which does add that x_0 term), which we will briefly call $\mathcal{L}$, is defined

$$\mathcal{L} : x = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \mapsto \theta^T \tilde{x} = \begin{pmatrix} \theta_0 & \dots & \theta_d \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{pmatrix}$$

It basically means that we first attach the $x_0 = 1$ term to create a $(d + 1)$-dimensional vector, and then we dot product it with θ. This definition naturally constructs the affine hyperplane of $\mathbb{R}^d$, defined

$$H \equiv \{x \in \mathbb{R}^d \mid \mathcal{L}(x) = 0\}$$

If d-dimensional x is in $H \subset \mathbb{R}^d$, then $\mathcal{L}(x) = 0$. But this output 0 is inputted into the logistic function

$$g(0) = \frac{1}{1 + e^{-0}} = \frac{1}{2}$$

This means that set of points in H represents the input values where the regression algorithm will output exactly $(0.5, 0.5)$. We can construct the quotient space

$$\mathbb{R}^d / H$$

and easily visualize the gradient, which lightens and darkens as we move from hyperplane to hyperplane. Now, given that

$$h_\theta(x) \equiv \frac{1}{1 + e^{-\theta^T x}}$$

(or really for any function $h_\theta : \mathbb{R}^{d+1} \longrightarrow [0, 1]$) such that

$$\begin{aligned} \mathbb{P}(y = 1 \mid x; \theta) &= h_\theta(x) \\ \mathbb{P}(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

which can be written more compactly as below. Remember, the notation means ¡i¿the probability of y given x and where θ is a given. ¡/i¿

$$p(y \mid x; \theta) = \big(h_\theta(x)\big)^y \, \big(1 - h_\theta(x)\big)^{1-y}$$

Assuming that the n training samples were generated independently, we can then write down the likelihood of the parameters as

$$\begin{aligned} L(\theta) &= p(y \mid X; \theta) \\ &= \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^{n} \big(h_\theta(x^{(i)})\big)^{y^{(i)}} \big(1 - h_\theta(x^{(i)})\big)^{1-y^{(i)}} \end{aligned}$$

As before, it is simpler to maximize the log likelihood, which is known as the

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \end{aligned}$$

But since there is no negative sign in front of the whole summation, we can simply just use gradient ascent to maximize l (The subscript θ on ∇ is there to clarify that the gradient is computed with respect to the θ, not X).

$$\theta = \theta + \nabla_\theta l(\theta)$$

But since $g'(z) = g(z)\,(1 - g(z))$, we can derive the gradient of $l : \mathbb{R}^{d+1} \longrightarrow \mathbb{R}$ with its partials

$$\begin{aligned} \frac{\partial}{\partial \theta_j} l(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= \big(y(1 - g(\theta^T x)\big) - (1 - y) g(\theta^T x)\big) x_j \\ &= \big(y - h_\theta(x)\big) x_j \end{aligned}$$

Therefore, the stochastic gradient ascent rule is reduced to (vector and coordinate form)

$$\theta = \theta + \alpha\big(y^{(i)} - h_\theta(x^{(i)}\big)\, x^{(i)} \iff \theta_j = \theta_j + \alpha\big(y^{(i)} - h_\theta(x^{(i)})\big)\, x_j^{(i)}$$

4 Generalized Linear Models

Remember the linear model looked like this, where we use the conventional β notation to represent parameters.

$$Y = X^T \beta + \epsilon, \quad \epsilon \sim N(0, \sigma^2 I)$$

which implies that $Y \mid X \sim N(X^T \beta, \sigma^2 I)$. Basically, given x, I assume some distribution of Y, and the value of x will help me guess what the mean of this distribution is. Note that we in here assume that only the mean depends on X. I could potentially have something crazy, like

$$Y \mid X \sim N(X^T \beta, (X^T \gamma)(X X^T + I))$$

where the covariance will depend on X, too, but in this case we only assume that that mean is dependent on X.

$$Y \mid X \sim N(\mu(X), \sigma^2 I)$$

where in the linear model, $\mu(X) = X^T \beta$. So, there are three assumptions we are making here:

1. $Y \mid X$ is Gaussian.
2. X only affects the mean of $Y \mid X$, written $\mathbb{E}[Y \mid X] = \mu(X)$.

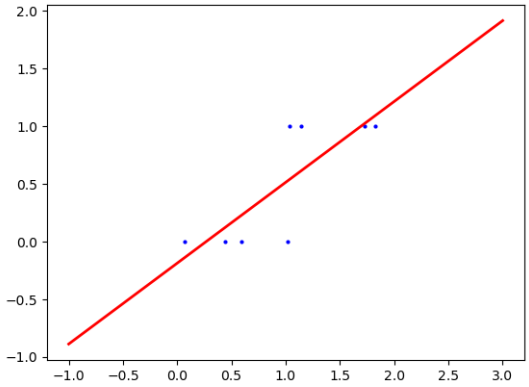


Figure 12: Fitting a linear model for Bernoulli random variables will predict a mean that is outside of $[0, 1]$ when getting new datapoints.

3. X affects the mean in a linear way, such that $\mu(X) = X^T\beta$.

So the two things we are trying to relax are:

1. **Random Component**: the response variable $Y \mid X$ is continuous and normally distributed with mean $\mu = \mu(X) = \mathbb{E}[Y \mid X]$.
2. **Link**: I have a link that explains the relationship between the X and the μ, and this relationship is $\mu(X) = X^T\beta$.

So when talking about GLMs, we are not changing the fact that we have a linear function $X \mapsto X^T\beta$. However, we are going to assume that $Y \mid X$ now comes from a broader **family of exponential distributions**. Second, we are going to assume that there exists some **link function** g

$$g(\mu(X)) = X^T\beta$$

Admittedly, this is not the most intuitive way to think about it, since we would like to have $\mu(X) = f(X^T\beta)$, but here we just decide to call $f = g^{-1}$. Therefore, if I want to give you a GLM, I just need to give you two things: the conditional distribution $Y \mid X$, which can be any distribution in the exponential family, and the link function g.

We really only need this link function due to compatibility reasons. Say that $Y \mid X \sim \text{Bern}(p)$. Then, $\mu(X) = \mathbb{E}[Y \mid X]$ always lives in $[0, 1]$, but $X^T\beta$ always lives in $\mathbb{R}$. We want our model to be realistic, and we can clearly see the problem shown in Figure 12. If $Y \mid X$ is some exponential distribution, then its support is always positive and so $\mu(X) > 0$. But if we stick to the old form of $\mu(X) = X^T\beta$, then $\text{Im}(\mu) = \mathbb{R}$, which is not realistic when we predict negative values. Let's take a couple examples:

Example 4.1. In the early stages of a disease epidemic, the rate at which new cases occur can often increase exponentially through time. Clearly, $\mu(X) = \mathbb{E}[Y \mid X]$ should be positive and we should have some sort of exponential trend. Hence, if $\mu(x)$ is the expected number of cases on data x, a model of the form

$$\mu(x) = \gamma \exp(\delta x)$$

seems appropriate, where γ and δ are simply scaling factors. Clearly, $\mu(X)$ is not of the form $f(X^T\beta)$. So what I do is to transform μ in such a way that I can get something that is linear.

$$\log(\mu(X)) = \log(\gamma) + \delta X$$

which is now linear in X, of form $\beta_0 + \beta_1 X$. This will have some effects, but this is what needs to be done to have a genearlized linear model. Note that what I did to μ was take the log of it, and so the link function

is $g = \log$, called the **log-link**. Now that we have chosen the g, we still need to choose what the conditional distribution $Y \mid X$ would be. This is determined by speaking with industry professionals, experience, and convenience. In this case, Y is a count, and since this must be a discrete distribution. Since it is not bounded above, we think Poisson.

Example 4.2. The rate of capture of preys, Y, by a hunting animal, tends to increase with increasing density of prey X, but eventually level off when the predator is catching as much as it can cope with. We want to find a perhaps concave function that levels off, and suitable model might be

$$\mu(X) = \frac{\alpha X}{h + X}$$

where α represents the maximum capture rate, and h represents the prey density at which the capture rate is half the maximum rate. Again, we must find some transformation g that turns this into a linear function of X, and what we can do it use the **reciprocal-link**.

$$\frac{1}{\mu(X)} = \frac{h + X}{\alpha X} = \frac{h}{\alpha}\frac{1}{X} + \frac{1}{\alpha}$$

The standard deviation of capture rate might be approximately proportional to the mean rate, suggesting the use of a Gamma distribution for the response.

Example 4.3. The Kyphosis data consist of measurements on 81 children following corrective spinal surgery. The binary response variable, Kyphosis, indicates the presence or absence of a postoperative deforming. The three covariates are: age of the child in months, number of the vertebrae involved in the operation, and the start of the range of the vertebrae involved. The response variable is binary so there is no choice: $Y \mid X$ is Bernoulli with expected value $\mu(X) \in (0, 1)$. We cannot write $\mu(X) = X^T\beta$ because the right hand side ranges through $\mathbb{R}$, and so we find an invertible function that squishes $\mathbb{R}$ to $(0, 1)$, and so we can choose basically any CDF.

For clarification, when writing a distribution like Bernoulli(p), or Binomial(n, p), Poisson(λ), or $N(\mu, \sigma^2)$, the hyperparameters that we usually work with we will denote as $\boldsymbol{\theta}$, and the space that this $\boldsymbol{\theta}$ lives in will denote Θ. For example, for the Bernoulli, $\Theta = [0, 1]$, and for Poisson, $\Theta = [0, +\infty)$.

Ultimately, a GLM consists of three steps:

1. The observed input X enters the model through a linear function $\beta^T X$.
2. The conditional mean of response, is represented as a function of the linear combination

$$\mathbb{E}[Y \mid X] = \mu = f(\beta^T X)$$

3. The observed response is drawn from an exponential family distribution with conditional mean μ.

4.1 Exponential Family

We can write the pdf of a distribution as a function of the input x and the hyperparameters θ, so we can write $P_\theta(x) = p(\theta, x)$. For now, let's think that both $x, \theta \in \mathbb{R}$. Think of all the functions that depend on θ and x. There are many of them, but we want θ and x to interact in a certain way. The way that I want them to interact with each other is that they are multiplied within an exponential term. Now clearly, this is not a very rich family, so we are just slapping some terms that depend only on θ and only on x.

$$p_\theta(x) = \exp(\theta x)h(x)c(\theta)$$

But now if $\theta \in \mathbb{R}^k$ and $x \in \mathbb{R}^q$, then we cannot simply take the product nor the inner product, but what we can do is map both of them into a space that has the same dimensions, so I can take the inner product. That is, let us map $\boldsymbol{\theta} \mapsto \boldsymbol{\eta}(\boldsymbol{\theta}) \in \mathbb{R}^k$ and $\mathbf{x} \mapsto \mathbf{T}(\mathbf{x}) \in \mathbb{R}^k$, and so our exponential distribution form would be generalized into something like

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp\left[\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \mathbf{T}(\mathbf{x})\right] h(\mathbf{x})c(\boldsymbol{\theta})$$

We can think of $c(\boldsymbol{\theta})$ as the normalizing term that allows us to integrate the pdf to 1.

$$\int_{\mathcal{X}} p_{\boldsymbol{\theta}}(\mathbf{x}) = c(\boldsymbol{\theta}) \int \exp\big[\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \mathbf{T}(\mathbf{x})\big] h(\mathbf{x})\, d\mathbf{x}$$

We can just push the $c(\boldsymbol{\theta})$ term into the exponential by letting $c(\boldsymbol{\theta}) = e^{-\log(c(\boldsymbol{\theta}))^{-1}}$ to get our definition.

Definition 4.1 (Exponential Family)**.** A **k-parameter exponential family** is a family of distributions with pdf/pmf of the form

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp\big[\boldsymbol{\eta}(\boldsymbol{\theta}) \cdot \mathbf{T}(\mathbf{x}) - B(\boldsymbol{\theta})\big] h(\mathbf{x})$$

The h term, as we will see, will not matter in our maximum likelihood estimation, so we keep it outside the exponential.

1. $\boldsymbol{\eta}$ is called the **canonical parameter**. Given a distribution parameterized by the regular hyperparameters $\boldsymbol{\theta}$, we would like to parameterize it in a different way $\boldsymbol{\eta}$ under the function $\boldsymbol{\eta} : \Theta \to \mathbb{R}$
2. $\mathbf{T}(\mathbf{x})$ is called the **sufficient statistic**.
3. $h(\mathbf{x})$ is a nonnegative scalar function.
4. $B(\boldsymbol{\theta})$ is the normalizing factor.

Let's look at some examples.

Example 4.4 (Gaussian)**.** If we put the coefficient into the exponential and expand the square term, we get

$$p_\theta(x) = \exp\left(\frac{\mu}{\sigma^2} \cdot x - \frac{1}{2\sigma^2} \cdot x^2 - \frac{\mu^2}{2\sigma^2} - \log(\sigma\sqrt{2\pi})\right)$$

where

$$\boldsymbol{\eta}(\boldsymbol{\theta}) = \begin{pmatrix} \mu/\sigma^2 \\ -1/2\sigma^2 \end{pmatrix},\ T(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix},\ B(\theta) = \frac{\mu^2}{2\sigma^2} + \log(\sigma\sqrt{2\pi}),\ h(x) = 1$$

This is not a unique representation since we can take the $\log(\sqrt{2\pi})$ out of the exponential, but why bother to do this when we can just stuff everything into B and keep h simple.

Example 4.5 (Gaussian with Known Variance)**.** If we have known variance, we can write the Gaussian pdf as

$$p_\theta(x) = \exp\left[\frac{\mu}{\sigma} \cdot \frac{x}{\sigma} - \frac{\mu^2}{2\sigma^2}\right] \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{x^2/2\sigma^2}$$

where

$$\eta(\theta) = \frac{\mu}{\sigma},\ T(x) = \frac{x}{\sigma},\ B(\theta) = \frac{\mu^2}{2\sigma^2},\ h(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{x^2/2\sigma^2}$$

Example 4.6 (Bernoulli)**.** The pmf of a Bernoulli with θ is

$$\begin{aligned} p_\theta(x) &= \theta^x (1-\theta)^{(1-x)} \\ &= \exp\big[x \log(\theta) + (1-x)\log(1-\theta)\big] \\ &= \exp\left(x \log\left[\frac{\theta}{1-\theta}\right] - \log\left[\frac{1}{1-\theta}\right]\right) \end{aligned}$$

where

$$\eta(\theta) = \log\left[\frac{\theta}{1-\theta}\right],\ T(x) = x,\ B(\theta) = \log\left[\frac{1}{1-\theta}\right],\ h(x) = 1$$

Example 4.7 (Binomial with Known Number of Trials)**.** We can transform a binomial with known N as

$$\begin{aligned} p_\theta(x) &= \binom{N}{x}\theta^x(1-\theta)^{1-x} \\ &= \exp\left[x\ln\left(\frac{\theta}{1-\theta}\right) + \ln(1-\theta)\right] \cdot \binom{N}{x} \end{aligned}$$

where

$$\eta(\theta) = \ln\left(\frac{\theta}{1-\theta}\right),\ T(x) = x,\ B(\theta) = \ln(1-\theta),\ h(x) = \binom{N}{x}$$

Example 4.8 (Poisson)**.** The pmf of Poisson with θ can be expanded

$$\begin{aligned} p_\theta &= \frac{\theta^{-x}}{x!}e^{-\theta} \\ &= \exp\big[-\theta + x\log(\theta) - \log(x!)\big] \\ &= \exp\big[x\log(\theta) - \theta\big]\,\frac{1}{x!} \end{aligned}$$

where

$$\eta(\theta) = \log(\theta),\ T(x) = x,\ B(\theta) = \theta,\ h(x) = \frac{1}{x!}$$

However, the uniform is not in here. In fact, any distribution that has a support that does not depend on the parameter is not an exponential distribution.

Let us now focus on one parameter families where $\theta \in \Theta \subset \mathbb{R}$, which do not include the Gaussian (with unknown mean and variance, Gamma, multinomial, etc.), which has a pdf written in the form

$$p_\theta(x) = \exp\big[\eta(\theta)\,T(x) - B(\theta)\big]\,h(x)$$

4.1.1 Canonical Exponential Family

Now a common strategy in statistical analysis is to reparamaterize a probability distribution. Suppose a family of probability distributions $\{P_\theta\}$ is parameterized by $\theta \in \Theta \subset \mathbb{R}$. If we have an invertible function $\eta : \Theta \to \mathcal{T} \subset \mathbb{R}$, then we can paramaterize the same family with η rather than θ, with no loss of information. Typically, it is the case that η is invertible for exponential families, so we can just reparameterize the whole pdf and write

$$p_\eta(x) = \exp\big[\eta\,T(x) - \phi(\eta)\big]\,h(x)$$

where $\phi = B \circ \eta^{-1}$.

Definition 4.2 (Canonical One-Parameter Exponential Family)**.** A family of distributions is said to be in **canonical one-parameter exponential family** if its density is of form

$$p_\eta(x) = \exp\big[\eta\,T(x) - \phi(\eta)\big]\,h(x)$$

which is a subfamily of the exponential family. The function ψ is called the **cumulant generating function**.

Before we move on, let us just provide a few examples.

Example 4.9 (Poisson)**.** The Poisson can be represented as

$$p_\theta(x) = \exp\big[x\log\theta - \theta\big]\,\frac{1}{x!}$$

Now let $\eta = \log\theta \implies \theta = e^\eta$. So, we can reparamaterize the density as

$$p_\eta(x) = \exp\big[x\eta - e^\eta\big]\,\frac{1}{x!}$$

where $P_\eta = \text{Poisson}(e^\eta)$ for $\eta \in \mathcal{T} = \mathbb{R}$, compared to $P_\theta = \text{Poisson}(\theta)$ for $\theta \in \Theta = \mathbb{R}^+$.

Example 4.10 (Gaussian)**.** Recall that the Gaussian with known parameter σ^2 and unknown $\theta = \mu$ is in the exponential family, since we can expand it as

$$p_\theta(x) = \exp\left[\frac{\mu}{\sigma^2} \cdot x - \frac{\mu^2}{2\sigma^2}\right] \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{x^2/2\sigma^2}$$

We can perform the change of parameter $\eta = \mu^2/2\sigma^2 \implies \mu = \sigma^2\eta$, and substituting this in will give the canonical representation

$$p_\eta(x) = \exp\left[\eta x - \frac{\sigma^2\eta^2}{2}\right] \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{x^2/2\sigma^2}$$

where now $P_\eta = N(\sigma^2\eta, \sigma^2)$ for $\eta \in \mathcal{T} = \mathbb{R}$, compared to $P_\theta = N(\theta, \sigma^2)$ for $\theta \in \Theta = \mathbb{R}$, which is basically the same space.

Example 4.11 (Bernoulli)**.** The Bernoulli has an exponential form of

$$p_\theta(x) = \exp\left[x \log\left(\frac{\theta}{1-\theta}\right) + \log(1-\theta)\right]$$

Now setting $\eta = \log\left(\frac{\theta}{1-\theta}\right) \implies \theta = \frac{1}{1+e^{-\eta}}$, and so $B(\theta) = -\log(1-\theta) = -\log\left(\frac{e^{-\eta}}{1+e^{-\eta}}\right) = \log(1+e^\eta) = \psi(\eta)$, and so the canonical paramaterization is

$$p_\eta(x) = \exp\left[x\eta - \log(1+e^\eta)\right]$$

We present two useful properties of the exponential family.

Theorem 4.1 (Moments)**.** Let random variable X be in the canonical exponential family P_η

$$p_\eta(x) = e^{\eta T(x) - \psi(\eta)} h(x)$$

Then, the expectation and variance are encoded in the cumulant generating function in the following way

$$\mathbb{E}[T(X)] = \psi'(\eta) \qquad \text{Var}[T(X)] = \psi''(\eta)$$

Proof. ■

Example 4.12. We show that this is consistent with the Poisson, normal, and Bernoulli distributions.

1. In the Poisson, $\psi(\eta) = e^\eta$, and so $\psi'(\eta) = e^\eta = \theta = \mathbb{E}[X]$. Taking the second derivative gives $\psi''(\eta) = e^\eta = \theta = \text{Var}[X]$, too.
2. In the Normal with known variance σ^2, we have $\psi(\eta) = \frac{1}{2}\sigma^2\eta^2$. So
$$\mathbb{E}[X] = \psi'(\eta) = \sigma^2\eta = \mu$$
$$\text{Var}[X] = \psi''(\eta) = \sigma^2$$
3. In the Bernoulli, we have $\psi(\eta) = \log(1+e^{-\eta})$. Therefore,
$$\mathbb{E}[X] = \psi'(\eta) = \frac{x^\eta}{1+x^\eta} = \frac{1}{1+e^{-\eta}} = \theta$$
$$\text{Var}[X] = \psi''(\eta) = -\left(\frac{1}{1+e^{-\eta}}\right)^2 e^{-\eta} \cdot -1 = \theta^2 \cdot \frac{1-\theta}{\theta} = \theta(1-\theta)$$

Theorem 4.2 (Convexity)**.** Consider a canonical exponential family with density

$$p_\eta(x) = e^{\eta T(x) - \psi(\eta)} h(x)$$

and natural parameter space $\mathcal{T}$. Then, the set $\mathcal{T}$ is convex, and the cumulant generating function ψ is convex on $\mathcal{T}$.

Proof. This can be proven using Holder's inequality. However, from the theorem above, note that $\text{Var}[T(X)] = \psi''(\eta)$ must be positive since we are talking about variance. This implies that the second derivative of ψ is positive, and therefore must be convex. ■

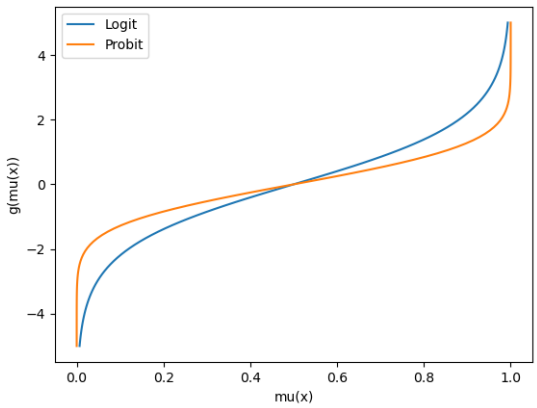


Figure 13: Logit and Probit Functions

4.2 Link Functions

Now let's go back to GLMs. In linear models, we said that the conditional expectation of Y given $X = \mathbf{x}$ must be a linear function in x

$$\mathbb{E}[Y \mid X = \mathbf{x}] = \mu(\mathbf{x}) = \mathbf{x}^T\beta$$

But if the conditional distribution takes values in some subset of $\mathbb{R}$, such as $(0, 1)$, then it may not make sense to write this as a linear function, since $X^T\beta$ has an image spanning $\mathbb{R}$. So what we need is a link function that relates, i.e. transforms the restricted subset of μ, onto the real line, so that now you can express it of the form $X^T\beta$.

$$g\big(\mu(X)\big) = X^T\beta$$

Again, it is a bit more intuitive to talk about g^{-1}, which takes our $X^T\beta$ and transforms it to the values that I want, so we will talk about both of them simultaneously. If g is our link function, we want it to satisfy 3 requirements:

1. g is continuously differentiable
2. g is strictly increasing
3. $\text{Im}(g) = \mathbb{R}$, i.e. it spans the entire real line

This implies that g^{-1} exists, which is also continuously differentable and is strictly increasing.

Example 4.13. If I have a conditional distribution...

1. that is Poisson, then we want our μ to be positive, and so we need a link function $g : \mathbb{R}^+ \to \mathbb{R}$. One choice would be the logarithm
$$g(\mu(X)) = \log\big(\mu(X)\big) = X^T\beta$$
2. that is Bernoulli, then we want our μ to be in $(0, 1)$ and we need a link function $g : (0, 1) \to \mathbb{R}$. There are 2 natural choices, which may be the **logit** function
$$g(\mu(X)) = \log\left(\frac{\mu(X)}{1 - \mu(X)}\right) = X^T\beta$$
or the **probit** function
$$g(\mu(X)) = \Phi^{-1}\big(\mu(X)\big) = X^T\beta$$
where Φ is the CDF of a standard Gaussian. The two functions can be seen in Figure 13.

Now there are many choices of functions we can take. In fact, if μ lives in $(0, 1)$, then we can really just take our favorite distribution that has a density that is supported everywhere in $\mathbb{R}$ and take the inverse cdf as our link. So far, we have no reason to prefer one function to another, but in the next section, we will see that there are more natural choices.

4.2.1 Canonical Link Functions

Now let's summarize what we have. We assume that the conditional distribution $Y \mid X = x$ follows a distribution in the exponential family, which we can completely characterize by the cumulant generating function ψ. For different values of x, the conditional distribution will be paramaterized by different $\eta(x)$, and the resulting distribution P_η will have some mean $\mu(x)$, which is usually not the natural parameter η. Now, let's forget about our knowledge that $\psi'(\eta) = \mu$, but we know that there is some relationship between $\eta \leftrightarrow \mu$.

Given an x, I need to use the linear predictor $x^T\beta$ to predict $\mu(x)$, which can be done through the link function g.

$$g\big(\mu(x)\big) = x^T\beta$$

Now what would be a natural way of choosing this g? Note that our natural parameter η for this canonical family takes value on the entire real line. I must construct a function g that maps μ onto the entire real line, and so why not make it map to η. Therefore, we have

$$\eta(x) = g\big(\mu(x)\big) = x^T\beta$$

Definition 4.3 (Canonical Link)**.** The function g that links the mean μ to the canonical parameter θ is called the **canonical link**.

$$g(\mu) = \theta$$

Now using our knowledge that $\psi'(\eta) = \mu$, we can see that

$$g = (\psi')^{-1}$$

This is indeed a valid link function.

1. $\psi'' > 0$ since it models the variance, and so ψ' is strictly increasing and so $g = (\psi')^{-1}$ is also strictly increasing.
2. The domain of ψ' is the real line since it takes in the natural parameter η which exists over $\mathbb{R}$, so $\text{Im}(g) = \mathbb{R}$.

So, given our cumulant generating function ψ and our link function g, both satisfying

$$\psi'(\eta) = \mu \text{ and } g(\mu) = x^T\beta$$

we can combine them to get

$$(g \circ \psi')(\eta) = g(\mu) = x^T\beta$$

and so, even though the mean of the response variable is not linear with respect to x, the value of $(g \circ \psi')(\eta)$ is indeed linear. In fact, if we choose the canonical link, then the equation

$$\eta = x^T\beta$$

means that the natural parameter of our conditional distribution in the exponential family is linear with respect to x! From this we can find the conditional mean $\mu(x)$.

The reason we focus on canonical link functions is because, when the canonical link is used, the components of the model (the parameters of the linear predictor) have an additive effect on the response variable in the transformed (linked) scale, which makes the interpretation of the results easier. It's also worth noting that while using the canonical link function has some desirable properties, it is not always the best or only choice, and other link functions may be used if they provide a better fit for the data or make more sense in the context of the problem at hand.

Let us evaluate some canonical link functions.

Example 4.14. The Bernoulli has the canonical exponential form of

$$p_\eta(x) = \exp\left[x\eta - \log(1 + e^\eta)\right]$$

where $\eta = \log\left(\frac{\theta}{1-\theta}\right)$. Since we have prior knowledge that $\theta = \mu$ (i.e. the expectation of a Bernoulli is the hyperparameter θ itself), we have a function that maps $\mu \mapsto \eta$.

$$\eta = g(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$$

which gives us our result. We can also take the inverse of $\psi' = \frac{e^\eta}{1+e^\eta}$ to get our result

$$g(\mu) = (\psi')^{-1}(\mu) = \log\left(\frac{\mu}{1-\mu}\right)$$

4.3 Likelihood Optimization

Now let us have a bunch of data points $\{(x_n, y_n)\}_{n=1}^N$. By our model assumption, we know that the conditional distribution $Y \mid X = x_n$ is now of an exponential family with parameter $\eta_n = \eta(x_n)$ and density

$$p_{\eta_n}(y_n) = \exp\big[y_n\eta_n - \psi(\eta_n)\big]h(y_n)$$

Now we want to do likelihood optimization on β (not η or μ), and to do this, we must rewrite the density function in a way so that it depends on β. Given a link function g, note the following relationship between β and η:

$$\begin{aligned}\eta_n = \eta(x_n) &= (\psi')^{-1}(\mu(x_n)) \\ &= (\psi')^{-1}\big(g^{-1}(x_n^T\beta)\big) \\ &= h(x_n^T\beta)\end{aligned}$$

where for shorthand notation, we define $h \coloneqq (g \circ \psi')^{-1}$. Subtituting this into the above likelihood, taking the product of all N samples, and logarithming the equation gives us the following log likelihood to optimize over β.

$$\ell(\beta) = \log \prod_{n=1}^N p_{\eta_n}(y_n) = \sum_{n=1}^N y_n h(x_n^T\beta) - \psi(h(x_n^T\beta))$$

where we dropped the $h(y_n)$ term at the end since it is a constant and does not matter. If g was the canonical link, then h is the identity, and we should have a linear relationship between $\eta(x_n) = x_n^T\beta$. This means that the η_n reduces only to $x_n^T\beta$, which is much more simple to optimize.

$$\ell(\beta) = \log \prod_{n=1}^N p_{\eta_n}(y_n) = \sum_{n=1}^N y_n x_n^T\beta - \psi(x_n^T\beta)$$

Note that the first term is linear w.r.t β, and ψ is convex, so the entire sum must be concave w.r.t. β. With this, we can bring in some tools of convex optimization to solve.