lecture05_final_linear-regression-part-2

September 22, 2016

LaTeX command declarations here.

```
In [1]: %matplotlib inline
        from Lec05 import *
```

1 EECS 445: Machine Learning

1.1 Lecture 05: Linear Regression II

- Instructor: Jacob Abernethy and Jia Deng
- Date: September 21, 2016

1.2 Outline for this Lecture

- Overfitting
- Regularized Least Squares
- Maximum Likelihood Interpretation of Linear Regression

1.3 Reading List

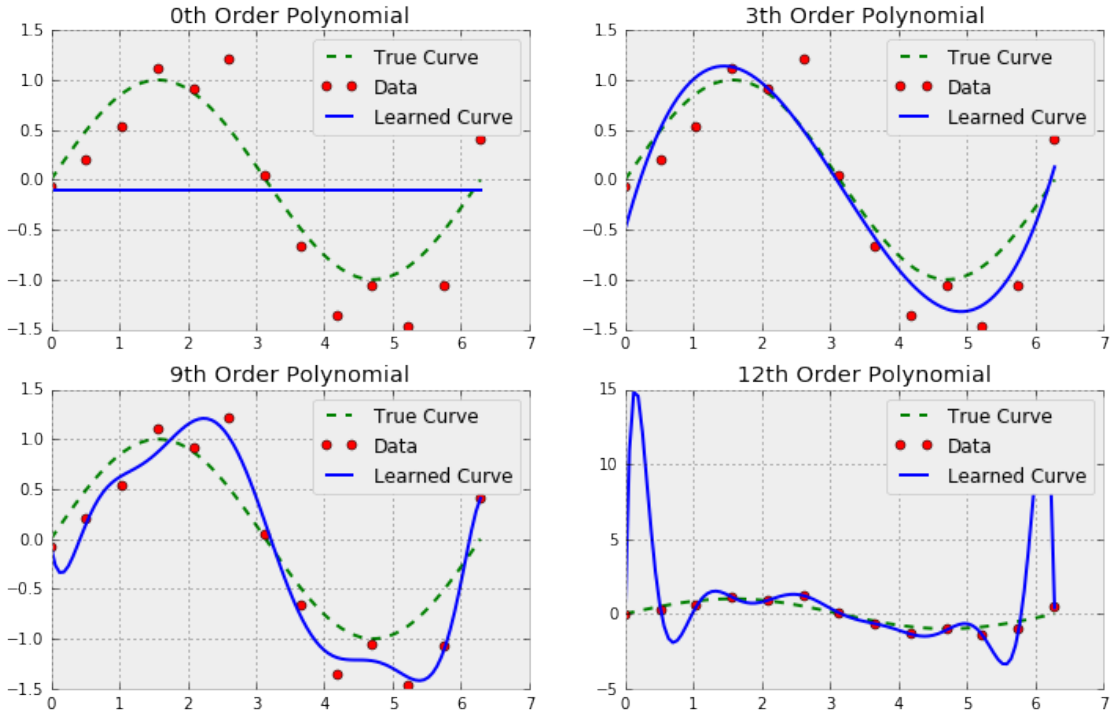
- Suggested:
  - **[PRML]**, §3.2: The Bias-Variance Decomposition
  - **[PRML]**, §3.3: Bayesian Linear Regression
- Optional:
  - **[MLAPP]**, Chapter 7: Linear Regression

In this lecture, we will first look at how degree of linear regression and sample dataset size will cause *overfitting* in linear regression. To deal with overfitting, *regularized least squares* will be introduced. When predicting the label of a new observation in linear regression, if we want to rely more on nearby traing data than distant training data, we will resort to *locally-weighted linear regression*. Finally, we will show regular linear regression and regularized linear regression can be interpreted from probablistic perspective each using *maximum likelihood estimation* and *maximum a posteriori estimation*.

1.4 Overfitting

1.4.1 Overfitting: Degree of Linear Regression

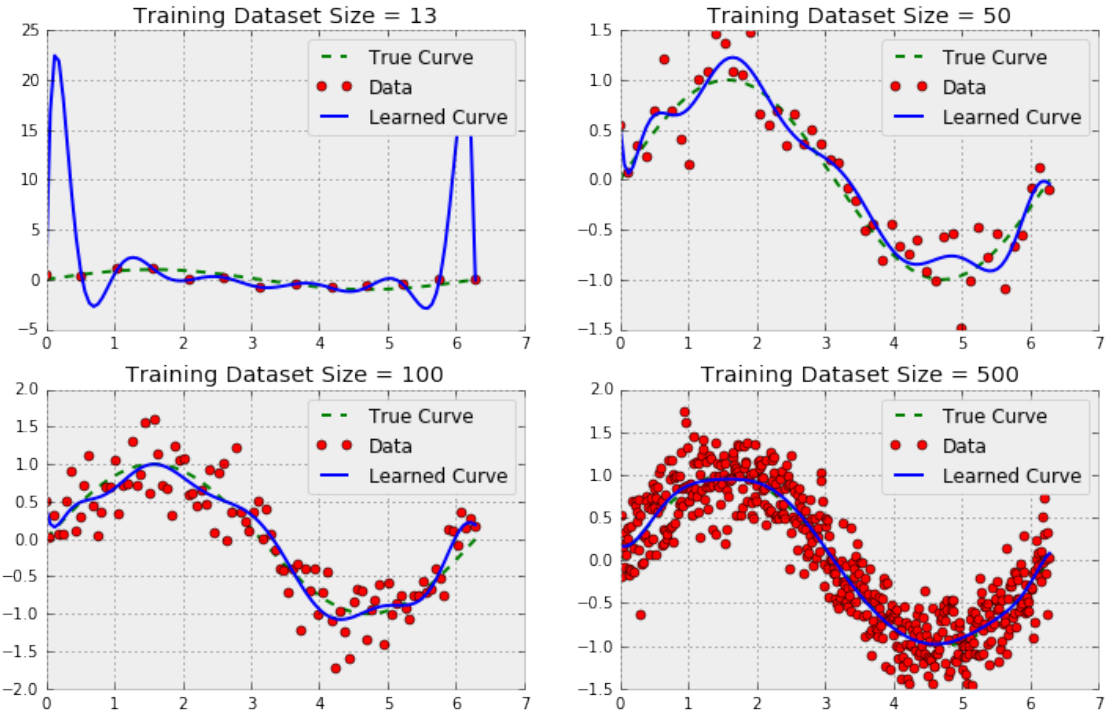
```
In [2]: regression_overfitting_degree(degree0=0, degree1=3,degree2=9,degree3=12)
```



Remark - In the above plots, we try to predict the true sinusoidal curve hidden in the data with polynomial degrees 0, 3, 9 and 12. - In the first plot (degree=0), we only get a horizontal line. The learned curve can neither fit the training data nor match true curve. This is called **Underfitting**. - In the second plot (degree=3), the predicted plot fits both data and true curve perfectly. This is a good degree for our setting. - In the third (degree=9) and fourth (degree=12) plots, as the degree increases, the traning data are fitted better but the learned curve deviates from the true curve further. This is called **Overfitting**. - Explanations of why degree could impact the predicted curve will come in the **Remark** later.

1.4.2 Overfitting: Dataset Size

```
In [3]: regression_overfitting_datasetsize(size0 = 13, size1 = 50, size2 = 100, siz
```

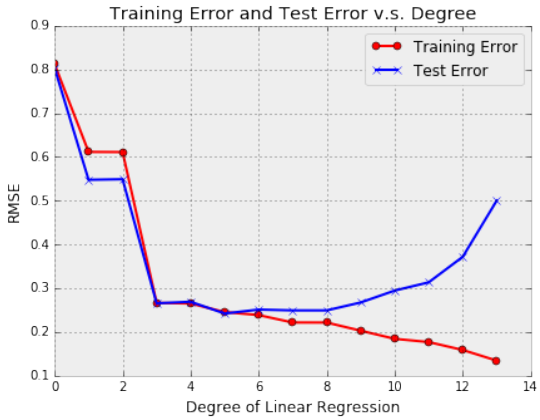


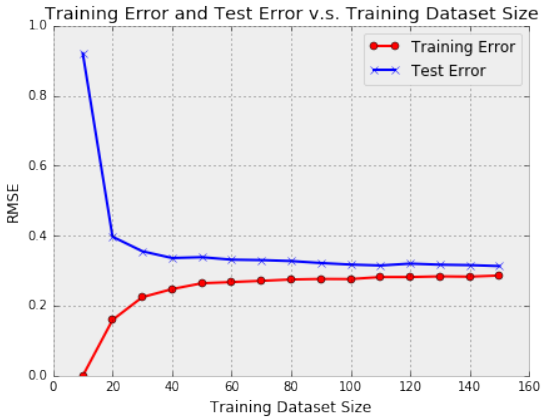
Remark - In the above plots, we try to predict the true sinusoidal curve hidden in the data with polynomial degree 12 and training dataset size 13, 50, 100 and 500. - In the first plot (size=13), **overfitting** occurs as we have seen just now. - Comparing the four plots altogether, we could see as the size increases, overfitting diminishes. And although more and more data points cannot be fitted by the learned curve (training error is becoming higher), but it matches the true curve better (test error will be smaller). - Explanations of why training data size impact the predicted curve will come in the **Remark** later.

1.4.3 Overfitting: Overall Performance

- On the left plot, we fix the dataset size and vary the polynomial degree
- On the right plot, we fix the polynomial degree and vary the dataset size

```
In [4]: regression_overfitting_curve()
```





Remark - The plot below is the **root mean squared error (RMSE)** of training dataset and test dataset with respect to different degree and dataset size. - NOTE: For simplicity of presentation, we divided the dataset into training set and test set. However, it's not legitimate to find the optimal hyperparameter based on the test set. We will talk about legitimate ways of doing this when we cover model selection and cross-validation. - Combining the last 10 plots, we could have: - **Degree** - When degree is really small, the regressor is not powerful enough (i.e. degree is small) to learn the underlying true model. At this time, underfitting occurs. Both training error and test error are high. - As degree increases, regressor become more powerful enough to roughly learn the true model but not that powerful to also take the noise into considerations. At this time, underfitting reduces. Both training error and test error will be smaller. - As degree further increases, regressor become so powerful that it could fit most of the data in training dataset perfectly. And since the data are noisy, the learned model actually deviates from the true model. At this time, overfitting occurs. Training error could becomes very small (even 0 sometimes), while test error increases. - **Dataset Size** - When training dataset size is small, a powerful regressor (i.e. degree is high) can fit every single sample in training dataset. Therefore, noise is also considered. This is equivalent to the ending zone of the plot (left) with respect degree. - As training dataset size increases, since the power of regressor is finite (degree is fixed), regressor starts to fail to fit every single sample. Instead it seeks to learn a curve such that samples will fall on both sides equally. Since the noise are assumed to 0 mean Gaussian noise which is symmetric with respect to 0, this is actually close to the underlying true curve. - The training error and test error will converge to 0.3 which is the variance of noise (you could check this by examining the Python code). This is not a coincidence and can be derived. We will cover this later when we study **bias-variance tradeoff** -

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - t_n)^2}$$

1.4.4 Rule of Thumb to Choose the Degree

- For a small number of datapoints, use a low degree

  - Otherwise, the model will overfit!

- As you obtain more data, you can gradually increase the degree
  - Add more features to represent more data
  - **Warning:** Your model is still limited by the finite amount of data available. The optimal model for finite data cannot be an infinite-dimensional polynomial!)

- Use **regularization** to control model complexity.

1.5 Regularized Linear Regression

1.5.1 Coefficients of Overfitting

- Before we move to regularized linear regression, let's first look at what happened to the coefficients $\vec{w}$ when there is overfitting.

```
In [5]: regression_overfitting_coeffs()
```

	M=0 (Underfitting)	M=3 (Good)	M=9 (Overfitting)	M=12 (Overfitting)
w_0	9.22491	7.66854	-0.240721	0.036502
w_1		-75.2974	6.36637	-1.282764
w_2		172.044	-69.1889	19.592658
w_3		-14.2807	397.481	-170.650848
w_4			-1290.02	934.785369
w_5			2328.31	-3348.827201
w_6			-2093.26	7896.286428
w_7			580.659	-11973.474257
w_8			247.134	10891.034834
w_9			-28.6505	-4862.568457
w_10				131.114412
w_11				582.180371
w_12				-28.827647

Remark - The table above corresponds to the coefficients (multiplied by 100 for better visualization) for different degrees. - We could see that when overfitting occurs, we get some really crazy and large numbers! - So one intuition to handle overfitting is to *penalize* large coefficients.

1.5.2 Regularized Least Squares: Objective Function

- Recall the objective function we minimizes in last lecture is

$$E(\vec{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\vec{w}^T\phi(\vec{x}_n) - t_n\right)^2$$

- To penalize the large coefficients, we will add one penalization/regularization term to it and minimize them altogether.

$$E(\vec{w}) = \underbrace{\frac{1}{2}\sum_{n=1}^{N}\left(\vec{w}^T\phi(\vec{x}_n) - t_n\right)^2}_{E_D(\vec{w})} + \underbrace{\boxed{\frac{\lambda}{2}\|\vec{w}\|^2}}_{E_W(\vec{w})}$$

of which $E_D(\vec{w})$ represents the term of sum of squared errors and $E_W(\vec{w})$ is the regularization term.

- λ is the regularization coefficient.
- If λ is large, $E_{\vec{W}}(\vec{w})$ will dominate the objective function. As a result we will focus more on minimizing $E_W(\vec{w})$ and the resulting solution $\vec{w}$ tends to have smaller norm and the $E_D(\vec{w})$ term will be larger.

1.5.3 Regularized Least Squares: Derivation

- Based on what we have derived in last lecture, we could write the objective function as

$$\begin{aligned} E(\vec{w}) &= \frac{1}{2}\sum_{n=1}^{N}\left(\vec{w}^T\phi(\vec{x}_n) - t_n\right)^2 + \frac{\lambda}{2}\|\vec{w}\|^2 \\ &= \frac{1}{2}\vec{w}^T\Phi^T\Phi\vec{w} - \vec{t}^T\Phi\vec{w} + \frac{1}{2}\vec{t}^T\vec{t} + \frac{\lambda}{2}\vec{w}^T\vec{w} \end{aligned}$$

- The gradient is

$$\begin{aligned} \nabla_{\vec{w}}E(\vec{w}) &= \Phi^T\Phi\vec{w} - \Phi^T\vec{t} + \lambda\vec{w} \\ &= (\Phi^T\Phi + \lambda I)\vec{w} - \Phi^T\vec{t} \end{aligned}$$

- Setting the gradient to 0, we will get the solution

$$\boxed{\hat{\vec{w}} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\vec{t}}$$

- In the solution to ordinary least squares which is $\hat{\vec{w}} = (\Phi^T\Phi)^{-1}\Phi^T\vec{t}$, we cannot guarantee $\Phi^T\Phi$ is invertible. But in regularized least squares, if $\lambda > 0$, $\Phi^T\Phi + \lambda I$ is always invertible.

1.5.4 Regularized Least Squares: Different Norms

- The ℓ^p norm of a vector $\vec{x}$ is defined as

$$\|\vec{x}\|_p = (\sum_{j=1}^{M}|x_j|^p)^{\frac{1}{p}}$$

- For the regularized least squares above, we used ℓ^2 norm. We could also use other ℓ^p norms for different regularizers and the objective function becomes

$$E(\vec{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\vec{w}^T\phi(\vec{x}_n) - t_n\right)^2 + \frac{\lambda}{2}\|\vec{w}\|_p^p$$

(Contour of Different p-norms)

- **Lasso** regularization (ℓ^1 regularization) tends to generate sparser solutions than **ridge** regression (ℓ^2 regularization)
(Image taken from here)

Remark - RSS is residual of sum of squares, which is the sum of squared errors $E_D(\vec{w})$ we use. - This plot is to illustrate intuitively why lasso has sparser solution than ridge regression. - Our objective function is

$$E(\vec{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\vec{w}^T\phi(\vec{x}_n) - t_n\right)^2 + \frac{\lambda}{2}\|\vec{w}\|_p^p$$

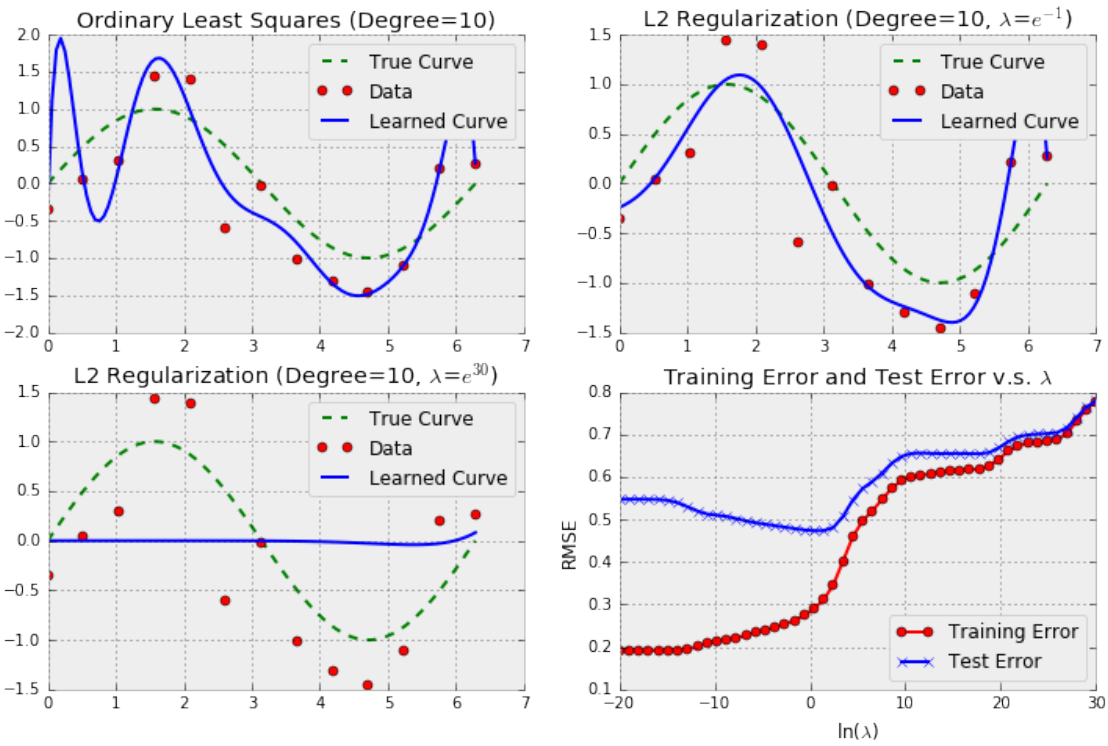
To illustrate, lets look at an *equivalent* constrained problem. (Not exactly equivalent, just for illustration purpose)

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\sum_{n=1}^{N}\left(\vec{w}^T\phi(\vec{x}_n) - t_n\right)^2 \\ \text{subject to} \quad & \frac{\lambda}{2}\|\vec{w}\|_p^p \leq C \end{aligned}$$

- The objective function $\frac{1}{2}\sum_{n=1}^{N}\left(\vec{w}^T\phi(\vec{x}_n) - t_n\right)^2$ is the red contours in the plots. The optimal solution without the constraint is $\hat{\vec{w}}_{OLS}$, which is just the solution to ordinary least squares. The farther we are away from $\hat{\vec{w}}_{OLS}$ the larger objective function will be. - The feasible area that satisfying the constraint is the cyan area in the plots. The solution must fall in these areas. For lasso, it's a diamond; for ridge regression, it's a circle. - As we increase the value of objective function, the contour will expand. The first time the contour *touch* the feasible cyan area, the touching point would be the optimal solution. This is because our solution should both be as close to $\hat{\vec{w}}_{OLS}$ as possible and fall in the feasible area. - Since lasso has diamond area with four corners, the contours tend to touch the corner first. And since the corner is on the axis where coordinate has at least one zero component, this guarantees the sparsity of solution. On the contrary, the circle area in ridge regression cannot give us this property.

1.5.5 Regularized Least Squares: Example

```
In [6]: regression_regularization_plot()
```



Remark - In the second plot, we can see that after the regularization term is added, the learned curve looks much more like the true curve than the learnd curve without regularization in the first plot. - However, in the third plot, when the coefficient λ for regularization is too large, the minimization will mostly focus on minimizing $\|\vec{w}\|$, thus deviating from the true curve in a great deal. (We will see the coefficients $\vec{w}$) are really small for this case in next slide) - In the fourth plot, as we increase λ, the training error is monotonically increasing because we are far away from the task of minimizing sum of squared error $E_D(\vec{w})$. As for the the test error, it has minimum value near $\lambda = 1$, which is the blancing point in the tradeoff between minimizing $E_D(\vec{w})$ and minimizing regularization term $E_W(\vec{w})$.

1.5.6 Regularized Least Squares: Coefficients

- Let's look at how the coefficients change after we add regularization

```
In [7]: regression_regularization_coeff()
```

	lambda=0	lambda=exp^1	lambda=exp^10
w_0	30.203406	13.701085	0.010221
w_1	7133.542582	13.267902	0.013419
w_2	-31022.107050	12.423422	0.020149
w_3	53507.324765	8.040454	0.029552
w_4	-48906.151251	0.865971	0.038200

```
w_5   26564.237381     -4.354079       0.034013
w_6   -9013.171136     -1.827607      -0.002813
w_7    1929.741748      2.147727      -0.054953
w_8    -253.351938     -0.613354       0.023119
w_9      18.620246      0.073751      -0.003275
w_10     -0.586531     -0.003284       0.000155
```

Remark - From the table above, we can see the regularization term has effectively constrained those huge coefficients. - However, when λ is large ($\lambda = e^{10}$), the coefficients are "over-regularized".

1.5.7 Regularized Least Squares: Summary

- Simple modification of linear regression
- ℓ^2 Regularization controls the tradeoff between *fitting error* and *complexity*.
  - Small ℓ^2 regularization results in complex models, but with risk of overfitting
  - Large ℓ^2 regularization results in simple models, but with risk of underfitting
- It is important to find an optimal regularization that *balances* between the two

1.6 Probablistic Interpretation of Least Squares Regression

- We have showed derived the solution to least squares regression by minimizing objective function. Now we will provide a probablistic perspective. Specifically, we will show the solution to **regular least squares** is just the **maximum likelihood** estimate of $\vec{w}$ and the solution to **regularized least squares** is the **Maximum a Posteriori** estimate.

1.6.1 Some Background

- Gaussian Distribution
$$\mathcal{N}(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{(x-\mu)^2}{2\sigma^2}\right]$$
- **Maximum Likelihood Estimation** and **Maximum a Posteriori Estimation (MAP)**
  - For distribution $t \sim p(t|\theta)$. θ is some unknown parameter (like mean or variance) to be estimated.
  - Given observation $\vec{t} = (t_1, t_2, \ldots, t_N)$,
    - The Maximum Likelihood Estimator is
$$\theta_{ML} = \arg\max \prod_{n=1}^{N} p(t_n|\theta)$$
    - If we have some prior knowledge about θ, the MAP estimator is
$$\theta_{MAP} = \arg\max \prod_{n=1}^{N} p(\theta|t_n) \quad \text{(Posteriori Probability of } \theta\text{)}$$

1.6.2 Maximum Likelihood Estimator $\vec{w}_{ML}$

- We assume the **signal+noise** model of single data $(\vec{x}, t)$ is

$$t = \vec{w}^T \phi(\vec{x}) + \epsilon$$
$$\epsilon \sim \mathcal{N}(0, \beta^{-1})$$

  of which $\vec{w}^T \phi(\vec{x})$ is the true model, ϵ is the perturbation/randomness.

- Since $\vec{w}^T \phi(\vec{x})$ is deterministic/non-random, we have

$$t \sim \mathcal{N}(\vec{w}^T \phi(\vec{x}), \beta^{-1})$$

- The **likelihood function** of t is just **probability density function (PDF)** of t

$$p(t|\vec{x}, \vec{w}, \beta) = \mathcal{N}(t|\vec{w}^T \phi(\vec{x}), \beta^{-1})$$

- For inputs $\mathcal{X} = (\vec{x}_1, \ldots, \vec{x}_n)$ and target values $\vec{t} = (t_1, \ldots, t_n)$, the data likelihood is

$$p(\vec{t}|\mathcal{X}, \vec{w}, \beta) = \prod_{n=1}^{N} p(t_n|\vec{x}_n, \vec{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\vec{w}^T \phi(\vec{x}_n), \beta^{-1})$$

- **Notation Clarification**
  - $p(t|x, w, \beta)$ it the PDF of t whose distribution is parameterized by $x, \vec{w}, \beta$.
  - $\mathcal{N}(\vec{w}^T \phi(\vec{x}), \beta^{-1})$ is Gaussian distribution with **mean** $\vec{w}^T \phi(\vec{x})$ and **variance** β^{-1}.
  - $\mathcal{N}(t|\vec{w}^T \phi(\vec{x}), \beta^{-1})$ is the PDF of $\vec{t}$ which has Gaussian distribution $\mathcal{N}(\vec{w}^T \phi(\vec{x}), \beta^{-1})$

1.6.3 Maximum Likelihood Estimator $\vec{w}_{ML}$

- **Main Idea of Maximum Likelihood Estimate**
  - Given $\{\vec{x}_n, t_n\}_{n=1}^N$, we want to find $\vec{w}_{ML}$ that maximizes data likelihood function

$$\vec{w}_{ML} = \arg\max p(\vec{t}|\mathcal{X}, \vec{w}, \beta) = \arg\max \prod_{n=1}^{N} \mathcal{N}(t_n|\vec{w}^T \phi(\vec{x}_n), \beta^{-1})$$

    and by derivation we will show $\vec{w}_{ML}$ is equivalent to the least squares solution $\hat{\vec{w}} = \Phi^\dagger \vec{t}$.

- **Intuition about Maximum Likelihood Estimation**
  - Finding maximum likelihood estimate $\vec{w}_{ML} = \arg\max p(\vec{t}|\mathcal{X}, \vec{w}, \beta)$ is just finding the parameter $\vec{w}$ under which for data $\mathcal{X} = (\vec{x}_1, \ldots, \vec{x}_n)$, observed $\vec{t} = (t_1, \ldots, t_n)$ is the most likely result to be generated among all possible $\vec{t}$.

1.6.4 Maximum Likelihood Estimator $\vec{w}_{ML}$: Derivation

- Single data likelihood is

$$p(t_n|\vec{x}_n, \vec{w}, \beta) = \mathcal{N}(t_n|\vec{w}^T\phi(\vec{x}_n), \beta^{-1}) = \frac{1}{\sqrt{2\pi\beta^{-1}}} \exp\left\{-\frac{1}{2\beta^{-1}}(t_n - \vec{w}^T\phi(x_n))^2\right\}$$

- Single data log-likelihood is

$$\ln p(t_n|\vec{x}_n, \vec{w}, \beta) = -\frac{1}{2}\ln 2\pi\beta^{-1} - \frac{\beta}{2}(\vec{w}^T\phi(x_n) - t_n)^2$$

  We use logarithm because maximizer of $f(x)$ is the same as maximizer of $\log f(x)$. Logarithm can convert product to summation which makes life easier.

- Complete data log-likelohood is

$$\begin{aligned}\ln p(\vec{t}|\mathcal{X}, \vec{w}, \beta) &= \ln\left[\prod_{n=1}^{N} p(t_n|\vec{x}_n, \vec{w}, \beta)\right] = \sum_{n=1}^{N} \ln p(t_n|\vec{x}_n, \vec{w}, \beta) \\ &= \sum_{n=1}^{N}\left[-\frac{1}{2}\ln 2\pi\beta^{-1} - \frac{\beta}{2}(\vec{w}^T\phi(x_n) - t_n)^2\right]\end{aligned}$$

- Maximum likelihood estimate $\vec{w}_{ML}$ is

$$\begin{aligned}\vec{w}_{ML} &= \arg\max_{\vec{w}} \ln p(\vec{t}|\mathcal{X}, \vec{w}, \beta) \\ &= \arg\max_{\vec{w}} \sum_{n=1}^{N}\left[-\frac{1}{2}\ln 2\pi\beta^{-1} - \frac{\beta}{2}(\vec{w}^T\phi(x_n) - t_n)^2\right] \\ &= \arg\max_{\vec{w}} \sum_{n=1}^{N}\left[-\frac{\beta}{2}(\vec{w}^T\phi(x_n) - t_n)^2\right] \\ &= \arg\min_{\vec{w}} \sum_{n=1}^{N}\left[(\vec{w}^T\phi(x_n) - t_n)^2\right]\end{aligned}$$

- Familiar? Recall the objective function we minimized in least squares is $E(\vec{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\vec{w}^T\phi(\vec{x}_n) - t_n\right)^2$, so we could conclude that

$$\boxed{\vec{w}_{ML} = \hat{\vec{w}}_{LS} = \Phi^\dagger\vec{t}}$$

1.6.5 MAP Estimator $\vec{w}_{MAP}$

- The **MAP estimator** is obtained by

$$\begin{aligned}
\vec{w}_{MAP} &= \arg\max p(\vec{w}|\vec{t}, \mathcal{X}, \beta) && \text{(Posteriori Probability)} \\
&= \arg\max \frac{p(\vec{w}, \vec{t}, \mathcal{X}, \beta)}{p(\mathcal{X}, t, \beta)} \\
&= \arg\max \frac{p(\vec{t}|\vec{w}, \mathcal{X}, \beta)p(\vec{w}, \mathcal{X}, \beta)}{p(\mathcal{X}, t, \beta)} \\
&= \arg\max p(\vec{t}|\vec{w}, \mathcal{X}, \beta)p(\vec{w}, \mathcal{X}, \beta) && (p(X, t, \beta) \text{ is irrelevant to } \vec{w}) \\
&= \arg\max p(\vec{t}|\vec{w}, \mathcal{X}, \beta)p(\vec{w})p(\mathcal{X})p(\beta) && \text{(Independence)} \\
&= \arg\max p(\vec{t}|\vec{w}, \mathcal{X}, \beta)p(\vec{w}) && \text{(Likelihood} \times \text{Prior)}
\end{aligned}$$

  We are just using **Bayes Theorem** for the above steps.
- The only difference from ML estimator is we have an extra term of PDF of $\vec{w}$. This is the **prior belief** of $\vec{w}$. Here, we assume,

$$\vec{w} \sim \mathcal{N}(\vec{0}, \alpha^{-1}I)$$

- ML vs. MAP
  - Maximum Likelihood: We know **nothing** about $\vec{w}$ initially and every $\vec{w}$ are equally likelihood
  - Maximum a Posteriori: We know **something** about about $\vec{w}$ initially and certain $\vec{w}$ are more likely (depending on **prior** $p(\vec{w})$). In another way, $\vec{w}$ are weighted.
- Assumption $\vec{w} \sim \mathcal{N}(\vec{0}, \alpha^{-1}I)$ makes sense because
  - In **regularized least squares**
    - We already know large coefficient $\vec{w}$ that may lead to overfitting should be avoided.
    - When we increase the regularization coefficient λ, the smaller $\|\vec{w}\|$ will be.
  - When use $\vec{w} \sim \mathcal{N}(\vec{0}, \alpha^{-1}I)$
    - $\vec{w} \sim \mathcal{N}(\vec{0}, \alpha^{-1}I)$ encodes the assumption that $\vec{w}$ with a smaller norm $\|\vec{w}\|$ is more "likely" than a $\vec{w}$ with a bigger norm.
    - When we increase α, variance is smaller, small $\|\vec{w}\|$ will be much more likely

1.6.6 MAP Estimator $\vec{w}_{MAP}$: Derivation

- $\vec{w} \sim \mathcal{N}(\vec{0}, \alpha^{-1}I)$ is **multivariate Gaussian** which has PDF

$$p(\vec{w}) = \frac{1}{\left(\sqrt{2\pi\alpha^{-1}}\right)^N} \exp\left\{-\frac{1}{2\alpha^{-1}} \sum_{n=1}^{N} w_n^2\right\}$$

- So the MAP estimator is

$$\begin{aligned}
\vec{w}_{MAP} &= \arg\max_{\vec{w}} p(\vec{t}|\vec{w}, \mathcal{X}, \beta)p(\vec{w}) = \arg\max_{\vec{w}} \left[\ln p(\vec{t}|\vec{w}, \mathcal{X}, \beta) + \ln p(\vec{w})\right] \\
&= \arg\min_{\vec{w}} \left[\sum_{n=1}^{N} \frac{\beta}{2}(\vec{w}^T\phi(x_n) - t_n)^2 + \frac{\alpha}{2}\sum_{n=1}^{N} w_n^2\right] \\
&= \arg\min_{\vec{w}} \left[\sum_{n=1}^{N} \frac{1}{2}(\vec{w}^T\phi(x_n) - t_n)^2 + \frac{1}{2}\frac{\alpha}{\beta}\|\vec{w}\|^2\right]
\end{aligned}$$

- Exactly the objective in regularized least squares! So

$$\boxed{\vec{w}_{MAP} = \hat{\vec{w}} = \left(\Phi^T\Phi + \frac{\alpha}{\beta}I\right)^{-1}\Phi^T\vec{t}}$$

** Remark **- Of the above expression, $\frac{\alpha}{\beta}$ corresponds to the regularization coefficient λ we used in previous regulazied least squares. -** Priors**: Represent prior beliefs about acceptable values for model parameters. - Example: In linear regression, ℓ^2 regularization can be interpreted as placing a Gaussian Prior on the regression coefficients. - All statistical models and machine learning algorithms make assumptions. - All reasoning is based on implicit assumptions. - A Bayesian will tell you that his prior is a way of explicitly stating those assumptions. - This can all get very philosophical, but... - Bayesian reasoning is best seen as a useful tool. - Many concepts in machine learning have Bayesian interpretations. - Choice of loss / error function, regularization, etc. - For a fully Bayesian take on machine learning, check out the** Murphy** textbook:

  - We will cover more about Bayesian reasoning when we move to **Bayesian Linear Regression**, a linear regression that is used for streaming data.