00_Introduction

September 10, 2020

1 00 - Course Introduction

- *Course Overview*: Introduction to machine learning and its role in variety of real-world problems in areas such as remote sensing and image processing.
- So, what is machine learning?

One definition of Machine Learning: Area of study to develop methods for computers to make (intelligent?) decisions without being explicitly programmed.

- Supervised Learning: Learning a mapping from input data to desired output values given labeled training data
- The above was a classification example. Each data point was classified into a discrete class (either conure or macaw). Regression is also commonly a supervised learning problem.
- The usual flow (but not always) for supervised learning is:
  - Training
  - Testing
- Subset of challenges include:
  - How do you know if you have *representative* training data?
  - How do you know if you extracted *good* features?
  - How do you know if you selected the *right* model?
  - How do you know if you trained the model *well*?
- Many of these issues are alleviated (not solved entirely, but helped significantly) with *LOTS AND LOTS* of data and good experimental design.
- Obtaining labeled training data is hard, expensive, time consuming and, in some cases, infeasible

from NEON: https://www.neonscience.org

- Unsupervised Learning: Learning structure from data without any labels
- There are many sub-areas in machine learning:
  - **Supervised Learning**: learn from labeled data
  - **Unsupervised Learning**: learn from unlabeled data
  - **Semi-supervised Learning**: some training data labeled, some not, use all during training

  - **Reinforcement Learning**: reinforcement based on action in an environment so to maximize/minimize a reward/penalty
  - **Multiple Instance Learning**: labels have a particular form of imprecision
  - **Active Learning**: obtaining labels online from a user/oracle in an intelligent fashion
  - **Transfer Learning**: having labels on a related problem and transferring it to the task of interest
  - **Structured Learning**
  - **Associative Learning**
  - *many more...*
- Most of us interact with systems that leverage machine learning regularly:
  - Alexa/Siri
  - Search result sorting
  - Recommendation systems (Netflix, Amazon, Twitter news feed, Facebook news feed)
  - Google map updates via street view / predict crowds on buses/trains/streets (https://ai.googleblog.com/2017/05/updating-google-maps-with-deep-learning.html; https://techcrunch.com/2019/06/27/google-maps-can-now-predict-how-crowded-your-bus-or-train-will-be/)
  - Facial recognition (iPhone FaceID, JetBlue http://mediaroom.jetblue.com/investor-relations/press-releases/2018/11-15-2018-184045420)
  - Hiring systems (https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G)
  - Learning-driven home automation systems (e.g., NEST)
  - Anywhere there is a computer making an automated decision, it could have been trained using machine learning
- What do you predict will be the capabilities and limitations of machine learning-driven technology in 20 years?
  - For comparison/reference, in 2000 (20 years ago), the state of technology was:
    - people are using mostly flip phones (iPod introduced 2001; iPhone introduced 2007)
    - Camera phones just recently introduced (first camera phone sold in 2000)
    - Netflix launched as a DVD rental service
    - AOL instant messenger was popular
    - Many had dial up internet at home
    - Napster was a popular peer-to-peer sharing service
    - CD-ROMs commonly used
    - USB thumb drives were not available (came on market late 2000)
    - Very few people were using Google (founded in 1998; people were more commonly using Yahoo, Altavista, etc)

Pre-requisite Material for Foundations of Machine Learning

- The following includes of list a topics that you should know to succeed in this course.
- This covers material from calculus, linear algebra, statistics, and programming in Python.
- Please review the following material carefully.
- If you do not feel confident in all of the following material, you may want to reconsider taking the course at this time.

Python Review

Some Python tutorials and references to go over are:

- Learn Python the hard way: https://learnpythonthehardway.org/python3/ (https://learnpythonthehardway.org/python3/)
- Python for absolute beginners: https://www.youtube.com/playlist?list=PLS1QulWo1RIaJECMeUT4LFwJ-ghgoSH6n (https://www.youtube.com/playlist?list=PLS1QulWo1RIaJECMeUT4LFwJ-ghgoSH6n)
- Python Docs tutorial: https://docs.python.org/3/tutorial/ (https://docs.python.org/3/tutorial/)
- Numpy quickstart tutorial: https://docs.scipy.org/doc/numpy/user/quickstart.html (https://docs.scipy.org/doc/numpy/user/quickstart.html)
- Python For Data Science - A Cheat Sheet For Beginners: https://www.datacamp.com/community/tutorials/python-data-science-cheat-sheet-basics (https://www.datacamp.com/community/tutorials/python-data-science-cheat-sheet-basics)
- Other Python Cheat Sheets: https://towardsdatascience.com/collecting-data-science-cheat-sheets-d2cdff092855 (https://towardsdatascience.com/collecting-data-science-cheat-sheets-d2cdff092855) (Credits to Karlijn Willems)

Calculus Review

Calculus topics to review include:

- Derivatives: https://www.youtube.com/watch?v=9vKqVkMQHKk&list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr&index=2 (https://www.youtube.com/watch?v=9vKqVkMQHKk&list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr&index=2)
- Chain rule and product rule: https://www.youtube.com/watch?v=YG15m2VwSjA&list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr&index=4 (https://www.youtube.com/watch?v=YG15m2VwSjA&list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr&index=4)
- Integration: https://www.youtube.com/watch?v=rfG8ce4nNh0&list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr&index=8 (https://www.youtube.com/watch?v=rfG8ce4nNh0&list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr&index=8)
- Taylor Series: https://www.youtube.com/watch?v=3d6DsjIBzJ4&list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr&index=11 (https://www.youtube.com/watch?v=3d6DsjIBzJ4&list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr&index=11)

Additional resources:

- Full 3Blue1Brown Calculus series: https://www.youtube.com/playlist?list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr (https://www.youtube.com/playlist?list=PLZHQObOWTQDMsr9K-rj53DwVRMYO3t5Yr)
- ML-CheatSheet for Calculus: https://ml-cheatsheet.readthedocs.io/en/latest/calculus.html (https://ml-cheatsheet.readthedocs.io/en/latest/calculus.html)

Linear Algebra Review

Topics and definitions to know include:

- Vector: https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=2&t=0s (https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=2&t=0s)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}$$

- Matrix:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & \cdots & x_{dn} \end{bmatrix} \in \mathcal{R}^{d\times n}$$

- Transpose operation: \begin{equation}

```
\mathbf{x}^T = \left[  x_1,  x_2 , \cdots , x_D \right]
\end{equation}
```

$$\left(\mathbf{A}^T\mathbf{B}\right)^T = \mathbf{B}^T\mathbf{A}$$

- Vector/Matrix scaling: Given a vector $\mathbf{x} \in \mathbb{R}^D$ and a scalar value a, *what is $a\mathbf{x}$? What does this operation do geometrically?*
- Vector/Matrix addition: Given $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^D$, *what is $\mathbf{x} + \mathbf{y}$? What is the geometric interpretation?*
- Vector/Matrix subtraction: Given $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^D$, *what is $\mathbf{x} - \mathbf{y}$? What is the geometric interpretation?*
- Inner product: $\mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x} = \sum_{i=1}^D x_i y_i$
- Outer product: $xy^\top = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}^\top = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_dy_1 & x_dy_2 & \cdots & x_dy_n \end{bmatrix} \in \mathcal{R}^{d\times n}.$
- Linear transformations: https://www.youtube.com/watch?v=kYB8IZa5AuE&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=3 (https://www.youtube.com/watch?v=kYB8IZa5AuE&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=3)
- Inverse: https://www.youtube.com/watch?v=uQhTuRlWMxw&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=7 (https://www.youtube.com/watch?v=uQhTuRlWMxw&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=7)
- L-p norm: Given a vector, $\mathbf{x}$ and a p-value, the l_p norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{d=1}^{D} |x_d|^p\right)^{\frac{1}{p}}$$

So, if $p = 2$, then the l_2 norm of a vector is:

$$\|\mathbf{x}\|_2 = \left(\sum_{d=1}^{D} |x_d|^2\right)^{\frac{1}{2}}$$

- Eigenvectors and Eigenvalues: https://www.youtube.com/watch?v=PFDu9oVAE-g&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=14 (https://www.youtube.com/watch?v=PFDu9oVAE-g&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&index=14)

In [3]:
```python
# Associated python code to review some of the concepts listed above.

import numpy as np
a = 2
x = np.array([[1],[2],[3]])
y = np.array([[4],[5],[6]])

#Print vector x
print('x:',x)

#Transpose vector x
print('x.T:',x.T)

#Scale vector x with scalar a
print('a*x:', a*x)

#Vector addition
print('x+y:', x+y)

#Vector subtraction
print('y-x:',y-x)
```

```
x: [[1]
 [2]
 [3]]
x.T: [[1 2 3]]
a*x: [[2]
 [4]
 [6]]
x+y: [[5]
 [7]
 [9]]
y-x: [[3]
 [3]
 [3]]
```

In [ ]:
```python
#Several ways to compute the inner product of vectors x and y in python/numpy

#First by using matrix multiplication operator '@' that numpy supports:
print(x.T@y)
print(y.T@x)

#Second with numpy.matmul function for matrix maultiplication:
print(np.matmul(x.T,y))
print(np.matmul(y.T,x))

#Third with numpy.inner function:
print(np.inner(x.T,y.T))

#Fourth with numpy.dot function, note that numpy.dot acts the same as '@ ' or 'np.matmul' for 2
D arrays:
print(x.T.dot(y))
print(y.T.dot(x))

#for 1D arrays, acts similar to numpy.inner function:
print(np.dot([1,2,3],[4,5,6]))
```

In [ ]:
```python
#Compute the outer product of vectors x and y
print(np.outer(x,y))
```

In [ ]:
```python
print(np.outer(x,y).T)
```

In [ ]:
```python
#Compute l-p norm for p = 2, 3, 1
x = np.array([1, 2, 3])
print(np.linalg.norm(x,ord=2))
print((x@x)**(1/2))
print(np.linalg.norm(x,ord=3))
print(np.linalg.norm(x,ord=1))
```

```
In [ ]: #Compute l-p norm for p = 2, 3, 1, 0
        x = np.array([-1, -2, -3])
        print(np.linalg.norm(x,ord=2))
        print((x@x)**(1/2))
        print(np.linalg.norm(x,ord=3))
        print(np.linalg.norm(x,ord=1))
        print(np.linalg.norm(x,ord=0))
```

```
In [ ]: #Compute l-p norm for p = 2, 3, 1, 0
        x = np.array([-1, 0, 3])
        print(np.linalg.norm(x,ord=2))
        print((x@x)**(1/2))
        print(np.linalg.norm(x,ord=3))
        print(np.linalg.norm(x,ord=1))
        print(np.linalg.norm(x,ord=0))
```

Note the notation:

- scalar values are unbolded (e.g., N, x)
- vectors are lower case and bolded (e.g., $\mathbf{x}$)
- matrices are uppercase and bolded (e.g., $\mathbf{A} \in \mathbb{R}^{D\times N}$)
- vectors are generally assumed to be column vectors (e.g., $\mathbf{x}^T = (x_1, \ldots, x_N)$ and $\mathbf{x} = (x_1, \ldots, x_N)^T$)

Additional reading and videos to review linear algebra concepts:

- Strang, Gilbert, et al. Introduction to linear algebra. Vol. 4. Wellesley, MA: Wellesley-Cambridge Press, 2009. Chapters 1-7
- Lay, David C. "Linear Algebra and its Applications, 3rd updated Edition." (2005).
- MITOpenCourseWare Linear Algebra: https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/ (https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/video-lectures/)
- 3Blue1Brown Linear Algebra Review: https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab (https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab)
- SciPy Cheat Sheet: Linear Algebra in Python: https://www.datacamp.com/community/blog/python-scipy-cheat-sheet (https://www.datacamp.com/community/blog/python-scipy-cheat-sheet)

Statistics Review

Topics and definitions to know include:

- Likelihood and Probability
- Expected Value: https://www.youtube.com/watch?v=j__Kredt7vY (https://www.youtube.com/watch?v=j__Kredt7vY)
- Variance and covariance: https://www.youtube.com/watch?v=ualmyZiPs9w (https://www.youtube.com/watch?v=ualmyZiPs9w)
- Random variables: https://www.youtube.com/watch?v=3v9w79NhsfI (https://www.youtube.com/watch?v=3v9w79NhsfI)
- Probability density functions: https://www.youtube.com/watch?v=Fvi9A_tEmXQ (https://www.youtube.com/watch?v=Fvi9A_tEmXQ)
- Marginal and conditional probability: https://www.youtube.com/watch?v=CAXQvTKP8sg (https://www.youtube.com/watch?v=CAXQvTKP8sg)
- Independence and conditional independence: https://www.youtube.com/watch?v=uzkc-qNVoOk (https://www.youtube.com/watch?v=uzkc-qNVoOk)
- Normal/Gaussian distribution: https://www.youtube.com/watch?v=hgtMWR3TFnY (https://www.youtube.com/watch?v=hgtMWR3TFnY)
- Central Limit Theorem: https://www.youtube.com/watch?v=JNm3M9cqWyc (https://www.youtube.com/watch?v=JNm3M9cqWyc)
- Bayes' Rule: https://www.youtube.com/watch?v=XQoLVl31ZfQ (https://www.youtube.com/watch?v=XQoLVl31ZfQ)

Additional Reading: Goodfellow, I. et al. "Deep Learning", MIT Press, 2016. Chapter 3: Probability and Information Theory, Pages 51-70. http://www.deeplearningbook.org/contents/prob.html (http://www.deeplearningbook.org/contents/prob.html)

01a_PolynomialCurveFitting

September 10, 2020

1 1. Polynomial Curve Fitting Example

- Lets begin by considering the polynomial curve fitting example in the first chapter of the text.
- Suppose we have a training set with N data samples, $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$ and corresponding desired outputs $\mathbf{t} = (t_1, t_2, \ldots, t_N)^T$ where sample x_i has the desired label t_i
- We generally organize data into *vectors* and *matrices*. Not only is it a common way to organize the data, but it allows us to easily apply linear algebraic operations during analysis.
- Suppose the data actually came from some unknown hidden function.
- In practice/application, we only have the training data and its corresponding desired values (both of which may be noisy). From this training data, we want to learn a mapping from input values x to the desired output values t.
- If we knew the hidden function, we would not need to learn the mapping - we would already know it. However, since we do not know the true underlying function, we need to do our best to estimate from the examples of input-output pairs that we have.
- We will learn (i.e., train a model to estimate) that mapping from the training data $\{\mathbf{x}, \mathbf{t}\}$.
- Then, when we are given test data, we can predict each test data point's t value using the mapping that we estimated.
- For this problem, we assume that the original data x is sufficient and appropriate (so, we do not need to preprocess or extract features). Then, we have completed steps 1 and 2 of the general approach listed in Section 0 above.
- Now we must assume a model. Lets assume a polynomial function as our model (following the example in the text):

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j \tag{1}$$

- Now we must *train* this model by estimating the unknown parameters ($\mathbf{w}$) that maps the training data, $\mathbf{x}$, to their desired values, $\mathbf{t}$, given some assumed value for M
  - We use an Objective Function, or a Cost/Loss Function, to train our model. See below.

- So, we have N discrete points from which to estimate $\mathbf{w}$. We can minimize the squared error to estimate the parameters:

$$\begin{aligned}
\arg\min_{\mathbf{w}} E(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}\left(y(x_n,\mathbf{w})-t_n\right)^2 && (2)\\
&= \frac{1}{2}\sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right)^2 && (3)
\end{aligned}$$

  - We square in the objective function instead of taking the absolute value BECAUSE squaring is easier to optimize (even though in some cases the absolute value is more desirable)

- Consider the following illustration of the error function: The red lines correspond to the error between the data and the functional approximation.

- We can write the error function compactly in matrix/vector form:

$$\begin{aligned}
E(\mathbf{w}) &= \frac{1}{2}\left([w_0,w_1,\ldots,w_M]\begin{bmatrix} 1 & 1 & \ldots & 1\\ x_1 & x_2 & \ldots & x_N\\ x_1^2 & x_2^2 & \ldots & x_N^2\\ \vdots & \vdots & \ddots & \vdots\\ x_1^M & x_2^M & \ldots & x_N^M\end{bmatrix} - [t_1,t_2,\ldots,t_N]\right)\\
&\quad \left([w_0,w_1,\ldots,w_M]\begin{bmatrix} 1 & 1 & \ldots & 1\\ x_1 & x_2 & \ldots & x_N\\ x_1^2 & x_2^2 & \ldots & x_N^2\\ \vdots & \vdots & \ddots & \vdots\\ x_1^M & x_2^M & \ldots & x_N^M\end{bmatrix} - [t_1,t_2,\ldots,t_N]\right)^T\\
&= \frac{1}{2}\left(\mathbf{w}^T\mathbf{X}^T-\mathbf{t}^T\right)\left(\mathbf{w}^T\mathbf{X}^T-\mathbf{t}^T\right)^T && (4)\\
&= \frac{1}{2}\left\|\mathbf{w}^T\mathbf{X}^T-\mathbf{t}^T\right\|_2^2 && (5)
\end{aligned}$$

REMEMBER: L-p norm: Given a vector, $\mathbf{x}$ and a p-value, the l_p norm is defined as:

$$\|\mathbf{x}\|_p = \left(\sum_{d=1}^{D}|x_d|^p\right)^{\frac{1}{p}} \qquad (6)$$

So, if $p=2$, then the l_2 norm of a vector is:

$$\|\mathbf{x}\|_2 = \left(\sum_{d=1}^{D}|x_d|^2\right)^{\frac{1}{2}} \qquad (7)$$

where

$$\begin{aligned}
\mathbf{X}^T &= \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_N \\ x_1^2 & x_2^2 & \dots & x_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^M & x_2^M & \dots & x_N^M \end{bmatrix} \qquad (8) \\
&= [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N] \qquad (9)
\end{aligned}$$

and

$$\mathbf{x}_i = \left[x_i^0, x_i^1, x_i^2, \dots, x_i^M\right]^T \qquad (10)$$

- So, we want $E(\mathbf{w})$ to be small. How do we solve for $\mathbf{w}$?
- We can take the derivative of the error function, set it to zero, and solve for the parameters. In general, this method does not guarantee that the parameters we estimate are minima of the error function (e.g., may be an inflection point, maxima). It is a necessary condition (but not sufficient). However, if the function is convex, then it will always find the global optima.
- How do we take the derivative of a function with respect to a vector?

$$\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}) = \left[\frac{\partial}{\partial x_1} f(\mathbf{x}), \frac{\partial}{\partial x_2} f(\mathbf{x}), \dots, \frac{\partial}{\partial x_n} f(\mathbf{x})\right]^\top \in \mathcal{R}^{n\times 1}.$$

- So, what would the derivative of $E(\mathbf{w})$ be with respect to $\mathbf{w}$?

$$\begin{aligned}
E(\mathbf{w}) &= \frac{1}{2}\sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right)^2 \qquad (11) \\
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= \left[\frac{\partial E(\mathbf{w})}{\partial w_0}, \frac{\partial E(\mathbf{w})}{\partial w_1}, \dots, \frac{\partial E(\mathbf{w})}{\partial w_M}\right]^T \qquad (12) \\
&= \left[\sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right) x_n^0, \sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right) x_n^1, \dots, \sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right) x_n^M\right]^T
\end{aligned}$$

- Similarly,

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \left[\frac{1}{2} 2\left(\mathbf{w}^T\mathbf{X}^T - \mathbf{t}^T\right)\mathbf{X}\right]^T \qquad (13)$$

where $\mathbf{X}^T = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_N \\ x_1^2 & x_2^2 & \dots & x_N^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^M & x_2^M & \dots & x_N^M \end{bmatrix}$.

- Then, we can set the derivative to zero and solve:

$$
\begin{aligned}
0 &= \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{t} && (14)\\
\mathbf{X}^T\mathbf{t} &= \mathbf{X}^T\mathbf{X}\mathbf{w} && (15)\\
\mathbf{w} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{t} && (16)
\end{aligned}
$$

- Let's look at this more closely, suppose M = 2 and N = 2:

$$
\begin{aligned}
E(\mathbf{w}) &= \frac{1}{2}\left([w_0, w_1, w_2]\begin{bmatrix} 1 & 1\\ x_1 & x_2\\ x_1^2 & x_2^2\end{bmatrix} - [t_1, t_2]\right)\left([w_0, w_1, w_2]\begin{bmatrix} 1 & 1\\ x_1 & x_2\\ x_1^2 & x_2^2\end{bmatrix} - [t_1, t_2]\right)^T\\
&= \frac{1}{2}\left(\left[w_0 + w_1x_1 + w_2x_1^2, w_0 + w_1x_2 + w_2x_2^2\right] - [t_1, t_2]\right)\\
&\quad \left(\left[w_0 + w_1x_1 + w_2x_1^2, w_0 + w_1x_2 + w_2x_2^2\right] - [t_1, t_2]\right)^T\\
&= \frac{1}{2}\left(\left[w_0 + w_1x_1 + w_2x_1^2 - t_1, w_0 + w_1x_2 + w_2x_2^2 - t_2\right]\right)\\
&\quad \left(\left[w_0 + w_1x_1 + w_2x_1^2 - t_1, w_0 + w_1x_2 + w_2x_2^2 - t_2\right]\right)^T\\
&= \frac{1}{2}\left(\left(w_0 + w_1x_1 + w_2x_1^2 - t_1\right)^2 + \left(w_0 + w_1x_2 + w_2x_2^2 - t_2\right)^2\right)
\end{aligned}
$$

- Then, let us work out the derivative with respect to the vector **w**

$$
\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= \left[\frac{\partial E(\mathbf{w})}{\partial w_0}, \frac{\partial E(\mathbf{w})}{\partial w_1}, \frac{\partial E(\mathbf{w})}{\partial w_2}\right]^T\\
&= \left[\frac{1}{2}\left(2\left(w_0 + w_1x_1 + w_2x_1^2 - t_1\right) + 2\left(w_0 + w_1x_2 + w_2x_2^2 - t_2\right)\right),\right.\\
&\quad \frac{1}{2}\left(2\left(w_0 + w_1x_1 + w_2x_1^2 - t_1\right)x_1 + 2\left(w_0 + w_1x_2 + w_2x_2^2 - t_2\right)x_2\right),\\
&\quad \left.\frac{1}{2}\left(2\left(w_0 + w_1x_1 + w_2x_1^2 - t_1\right)x_1^2 + 2\left(w_0 + w_1x_2 + w_2x_2^2 - t_2\right)x_2^2\right)\right]^T\\
&= \frac{1}{2}2\begin{bmatrix} 1 & 1\\ x_1 & x_2\\ x_1^2 & x_2^2\end{bmatrix}\left([w_0, w_1, w_2]\begin{bmatrix} 1 & 1\\ x_1 & x_2\\ x_1^2 & x_2^2\end{bmatrix} - [t_1, t_2]\right)^T && (17)
\end{aligned}
$$

Thus,

$$
\begin{aligned}
0 &= \left(\mathbf{w}^T\mathbf{X}^T - \mathbf{t}^T\right)\mathbf{X} && (18)\\
0 &= \mathbf{w}^T\mathbf{X}^T\mathbf{X} - \mathbf{t}^T\mathbf{X} && (19)\\
\mathbf{t}^T\mathbf{X} &= \mathbf{w}^T\mathbf{X}^T\mathbf{X} && (20)\\
\mathbf{t}^T\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1} &= \mathbf{w}^T\mathbf{X}^T\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1} && (21)\\
\mathbf{t}^T\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1} &= \mathbf{w}^T && (22)\\
\mathbf{w} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{t} && (23)\\
& && (24)
\end{aligned}
$$

1.1 Apply to data generated from a (noisy) sine curve

- Suppose our data actually came from: $t = \sin(2\pi x) + \epsilon$ where ϵ is Gaussian zero-mean random noise.
- The univariate Gaussian Distribution:

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \tag{25}$$

  - where

$$\mu \tag{26}$$

  = mean (location of peak),

$$\sigma^2 \tag{27}$$

  = variance (how wide Gaussian will be)

  -

$$\int_{-inf}^{inf} \mathcal{N}(x|\mu,\sigma^2)dx = 1 \tag{28}$$

- If the noise is zero-mean Gaussian distributed, it is like we are saying there is a Gaussian around the true curve:

$$t = y + \epsilon \tag{29}$$
$$\epsilon = t - y \tag{30}$$

where

$$\epsilon \sim \mathcal{N}(0,\sigma^2) \tag{31}$$

thus

$$\mathcal{N}(t-y|0,1) \propto \exp\left\{-\frac{1}{2}\frac{(t-y-0)^2}{1^2}\right\} \tag{32}$$
$$= \exp\left\{-\frac{1}{2}(t-y)^2\right\} \tag{33}$$
$$= \exp\left\{-E(\mathbf{w})\right\} \tag{34}$$

- So, the squared error objective function, $E(\mathbf{w})$, assumes Gaussian noise.
- Another way to look at it: t is distributed according to a Gaussian distribution with mean y
- Also, *What is the multivariate Gaussian distribution?*
- First, lets generate data from the *true* underlying function (which, in practice, we would not know)

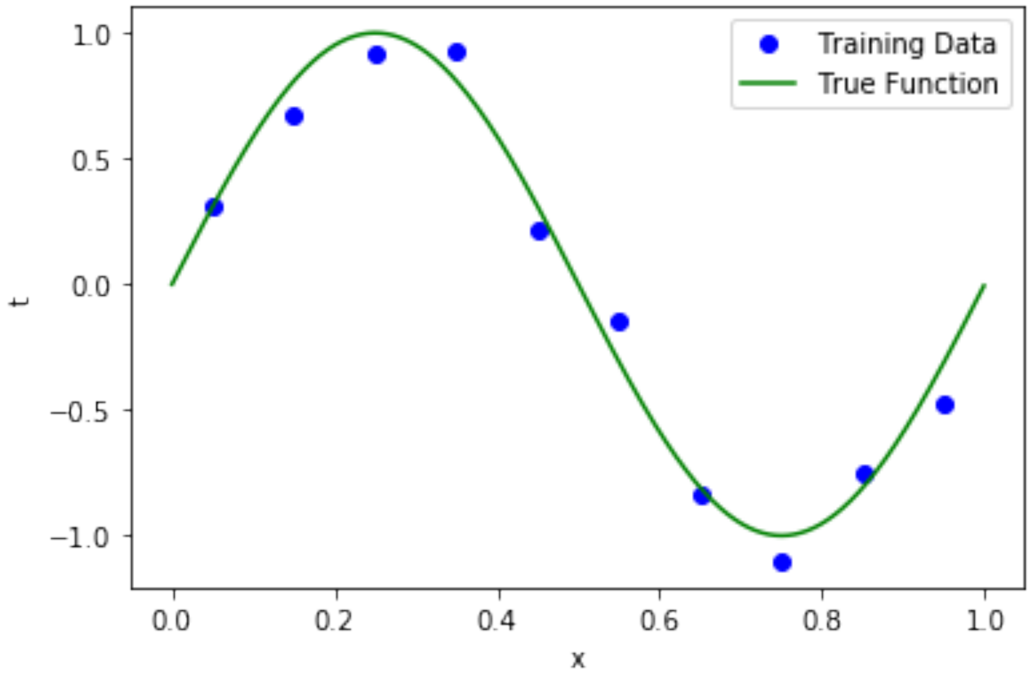
[7]:
```
# %load ../HelperCode/generateUniformData.py
import numpy as np
import math

def generateUniformData(N, l, u, gVar):
        '''generateUniformData(N, l, u, gVar): Generate N uniformly spaced data␣
↪points
    in the range [l,u) with zero-mean Gaussian random noise with standard␣
↪deviation gVar'''
        # x = np.random.uniform(l,u,N)
        step = (u-l)/(N)
        x = np.arange(l+step/2,u+step/2,step)
        e = np.random.normal(0,gVar,N)
        t = np.sin(2*math.pi*x) + e
        return x,t
```

- Lets plot this data and the underlying *true* function

[15]:
```
# %load ../HelperCode/plotData.py
import matplotlib.pyplot as plt
def plotData(x1,t1,x2,t2,x3=None,t3=None,legend=[]):

    #plot everything
    p1 = plt.plot(x1, t1, 'bo') #plot training data
    p2 = plt.plot(x2, t2, 'g') #plot true value
    if(x3 is not None):
        p3 = plt.plot(x3, t3, 'r')

    #add title, legend and axes labels
    plt.ylabel('t') #label x and y axes
    plt.xlabel('x')

    if(x3 is None):
        plt.legend((p1[0],p2[0]),legend)
    else:
        plt.legend((p1[0],p2[0],p3[0]),legend)
```

[9]:
```
import matplotlib.pyplot as plt
import textwrap
%matplotlib inline

l = 0
u = 1
N = 10
gVar = .1
data_uniform  = np.array(generateUniformData(N, l, u, gVar)).T

x1 = data_uniform[:,0]
t1 = data_uniform[:,1]
```

```
x2 = np.arange(l,u,0.001)  #get equally spaced points in the xrange
t2 = np.sin(2*math.pi*x2) #compute the true function value

fig = plt.figure()
plotData(x1, t1, x2, t2,legend=['Training Data', 'True Function'])
```

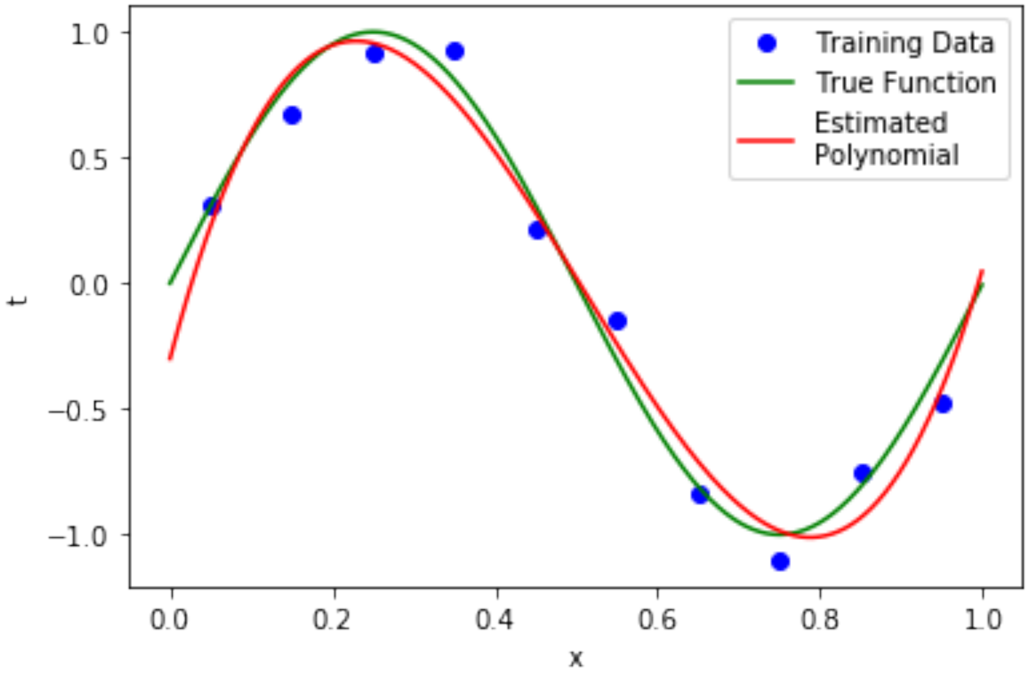


- Now lets fit the data using the polynomial curve fitting approach

[10]:
```
def fitdata(x,t,M):
        '''fitdata(x,t,M): Fit a polynomial of order M to the data␣
 ↪(x,t)'''
        #This needs to be filled in
        X = np.array([x**m for m in range(M+1)]).T
        w = np.linalg.inv(X.T@X)@X.T@t
        return w



M = 3
w = fitdata(x1,t1,M)
xrange = np.arange(l,u,0.001)  #get equally spaced points in the xrange
X = np.array([xrange**m for m in range(w.size)]).T
esty = X@w #compute the predicted value
plotData(x1,t1,x2,t2,xrange,esty,['Training Data', 'True Function',␣
 ↪'Estimated\nPolynomial'])
```

```
w
```

```
[10]: array([ -0.29972078,  12.22541022, -34.47274983,  22.60528008])
```



Lecture 3 - Linear Models for Regression with Basis Functions

In our **Polynomial Curve Fitting** example, the function is linear with respect to the parameters we are estimating. Thus, it is considered a *linear regression model*. (However, it is *non-linear* with respect to the input variable, x)

Input Space

Suppose we are given a training set comprising of N observations of $\mathbf{x}$, $\mathbf{x} = [x_1, x_2, \dots, x_N]^T$, and its corresponding desired outputs $\mathbf{t} = [t_1, t_2, \dots, t_N]^T$, where sample x_i has the desired label t_i. The input space is defined by the domain of $\mathbf{x}$.

- The polynomial curve fitting example can be rewritten as follows:

$$\begin{aligned} t \sim y(x, \mathbf{w}) &= w_0 + \sum_{j=1}^{M} w_j x^j \\ &= \sum_{j=0}^{M} w_j \phi_j(x) \end{aligned}$$

where

$$\phi_j(x) = x^j$$

- By modifying the function ϕ (known as a *basis function*), we can easily extend/modify the class of models being considered.

Linear Basis Model

The linear basis model for regression takes linear combinations of fixed nonlinear functions of the input variables

$$t \sim y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M} w_j \phi_j(\mathbf{x})$$

where $\mathbf{w} = [w_0, w_1, \dots, w_M]^T$ and $\mathbf{x} = [x_1, \dots, x_D]^T$

- For all data observations $\{x_i\}_{i=1}^N$ and using the basis mapping defined as $\boldsymbol{\phi}(x_i) = \begin{bmatrix} x_i^0 & x_i^1 & x_i^2 & \cdots & x_i^M \end{bmatrix}^T$, we can write the input data in a *matrix* form as:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^M \\ 1 & x_2 & x_2^2 & \cdots & x_2^M \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^M \end{bmatrix} = \begin{bmatrix} \boldsymbol{\phi}^T(x_1) \\ \boldsymbol{\phi}^T(x_2) \\ \vdots \\ \boldsymbol{\phi}^T(x_N) \end{bmatrix} \in \mathbb{R}^{N\times(M+1)}$$

where each row is a feature representation of a data point x_i.

Other **basis functions** include:

- Radial Basis functions (D = 1): $\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2)}{2s^2}\right\}$ where $x \in R^1$
- Radial Basis function (D > 1): $\phi_j(\mathbf{x}) = \exp\left\{-\frac{1}{2}(x-\boldsymbol{\mu}_j)^T\Sigma_j^{-1}(x-\boldsymbol{\mu}_j)\right\}$ where $\mathbf{x} \in R^D$, $\boldsymbol{\mu}_j \in R^D$ and $\boldsymbol{\Sigma}_j \in R^{D\times D}$
- Fourier Basis functions
- Wavelets Basis Functions

Feature Space

The domain of $\boldsymbol{\phi}(\mathbf{x})$ defines the **feature space**:

$$\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^{M+1}$$
$$\boldsymbol{\phi}(\mathbf{x}) \to [1, \phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_M(\mathbf{x})]$$

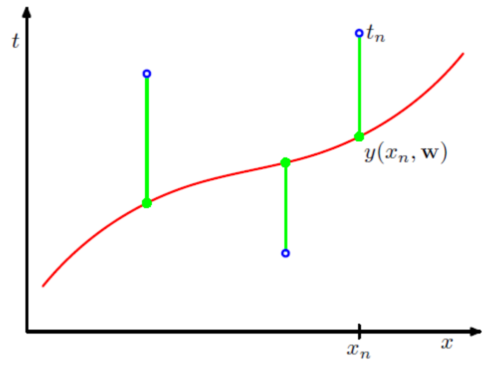
- When we use linear regression with respect to a set of (non-linear) basis functions, the regression model is linear in the *feature space* but non-linear in the input space.

Objective Function

We *fit* the polynomial regression model such that the *objective function* $E(\mathbf{w})$ is minimized:

$$\arg_{\mathbf{w}} \min E(\mathbf{w})$$

where $E(\mathbf{w}) = \frac{1}{2}||\boldsymbol{\Phi}\mathbf{w} - \mathbf{t}||_2^2$



- This error function is minimizing the (Euclidean) *distance* of every point to the curve.

We **optimize** $E(\mathbf{w})$ by finding the *optimal* set of parameters $\mathbf{w}^*$ that minimize the error function.

To do that, we **take the derivative of $E(\mathbf{w})$ with respect to the parameters $\mathbf{w}$**.

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \left[\frac{\partial E(\mathbf{w})}{\partial w_0}, \frac{\partial E(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial E(\mathbf{w})}{\partial w_M}\right]^T$$

- If we rewrite the objective function as:

$$\begin{aligned}
E(\mathbf{w}) &= \frac{1}{2}(\boldsymbol{\Phi}\mathbf{w} - \mathbf{t})^T (\boldsymbol{\Phi}\mathbf{w} - \mathbf{t}) \\
&= \frac{1}{2}\left(\mathbf{w}^T\boldsymbol{\Phi}^T - \mathbf{t}^T\right)(\boldsymbol{\Phi}\mathbf{w} - \mathbf{t}) \\
&= \frac{1}{2}\left(\mathbf{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\mathbf{w} - \mathbf{w}^T\boldsymbol{\Phi}^T\mathbf{t} - \mathbf{t}^T\boldsymbol{\Phi}\mathbf{w} + \mathbf{t}^T\mathbf{t}\right)
\end{aligned}$$

- Solving for $\mathbf{w}$, we find:

$$\begin{aligned}
\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} &= 0 \\
\frac{\partial}{\partial \mathbf{w}}\left[\frac{1}{2}\left(\mathbf{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\mathbf{w} - \mathbf{w}^T\boldsymbol{\Phi}^T\mathbf{t} - \mathbf{t}^T\boldsymbol{\Phi}\mathbf{w} + \mathbf{t}^T\mathbf{t}\right)\right] &= 0 \\
\frac{\partial}{\partial \mathbf{w}}\left[\left(\mathbf{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\mathbf{w} - \mathbf{w}^T\boldsymbol{\Phi}^T\mathbf{t} - \mathbf{t}^T\boldsymbol{\Phi}\mathbf{w} + \mathbf{t}^T\mathbf{t}\right)\right] &= 0 \\
(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\mathbf{w})^T + \mathbf{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} - (\boldsymbol{\Phi}^T\mathbf{t})^T - \mathbf{t}^T\boldsymbol{\Phi} &= 0 \\
\mathbf{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \mathbf{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} - \mathbf{t}^T\boldsymbol{\Phi} - \mathbf{t}^T\boldsymbol{\Phi} &= 0 \\
2\mathbf{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi} &= 2\mathbf{t}^T\boldsymbol{\Phi} \\
(\mathbf{w}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi})^T &= (\mathbf{t}^T\boldsymbol{\Phi})^T, \text{apply transpose on both sides} \\
\boldsymbol{\Phi}^T\boldsymbol{\Phi}\mathbf{w} &= \boldsymbol{\Phi}^T\mathbf{t} \\
\mathbf{w} &= \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}
\end{aligned}$$

Type *Markdown* and LaTeX: α^2

02_Overfitting

September 10, 2020

0.1 Overfitting/Overtraining

- In the polynomial curve fitting example, M is the *model order*.
- As M increases, there are more parameters (more w) to learn and, so, the model becomes more complex.

- As a model is more and more complex, it is more likely to *overfit* or *overtrain.* This essentially means it may "memorize" the input training data (including all of the training data's noise).

- Overfitting means that the performance of the model will likely decrease on unknown test data. Overfitting means that the "true" underlying model of the data is not estimated/learned but instead results in a poor representation that memorizes meaningless noise in the data.
- There are two common approaches to avoid overfitting:
  1. More data: As you have more and more data, it becomes more and more difficult to "memorize" the data and its noise. Often, more data translates to the ability to use a more complex model and avoid overfitting. However, generally, you need exponentially more data with increases to model complexity. So, there is a limit to how much this helps. If you have a very complex model, you need a huge training data set.
  2. Regularization: Regularization methods add a penalty term to the error function to discourage overfitting. These penalty terms encourage small values limiting the ability to overfit. (This is just a teaser. We will discuss this further in the future.)
- You can also *underfit* your data. When you underfit, your model complexity is not complex enough to model all of the complexities in your data.

0.2 Beer Foam Example

- Lets go through the Polynomial Curve fitting again with another example
- Obtained from: http://users.stat.ufl.edu/~winner/data/beer_foam.dat

Source: A. Leike (2002). "Demonstration of the Exponential Decay Law Using Beer Froth," European Journal of Physics, Vol. 23, #1, pp. 21-26

Description: Measurements of wet foam height and beer height at various time points for 3 brands of beer. Author fits exponential decay model: $H(t) = H(0)e^{-\lambda t}$

Variables/Columns:

Time from pour (seconds) 4-8

Erdinger Weissbier foam height (cm) 10-16

Augustinerbrau Munchen foam height (cm) 18-24

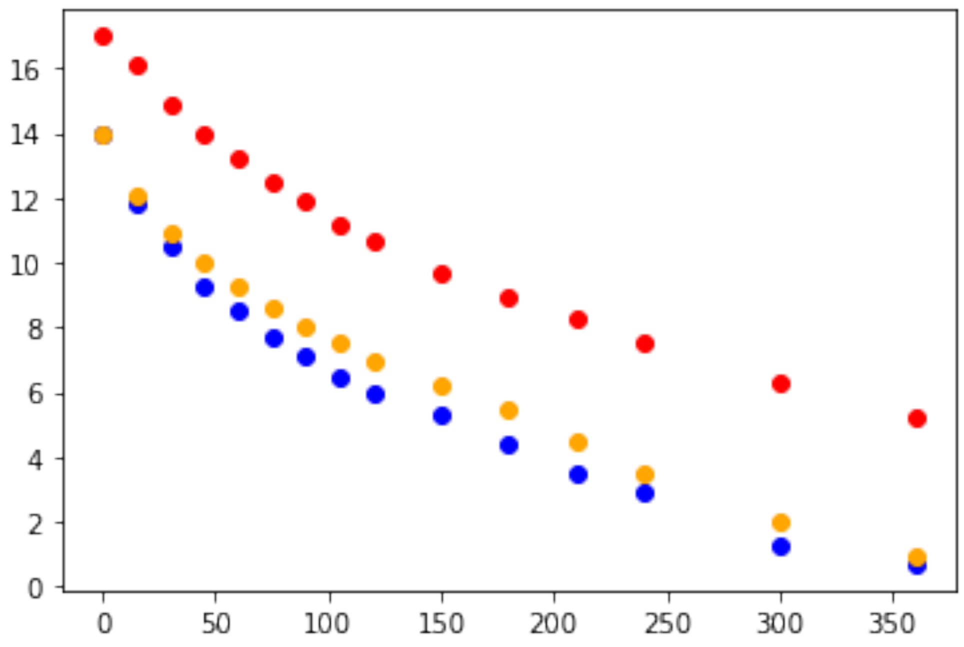
Budweiser foam height (cm) 26-32

[14]:
```
import numpy as np
# %load ../HelperCode/plotData.py
```

[15]:
```
def fitdata(x,t,M):
        '''fitdata(x,t,M): Fit a polynomial of order M to the data␣
 ↪(x,t)'''
        #This needs to be filled in
        X = np.array([x**m for m in range(M+1)]).T
        w = np.linalg.inv(X.T@X)@X.T@t
        return w
```

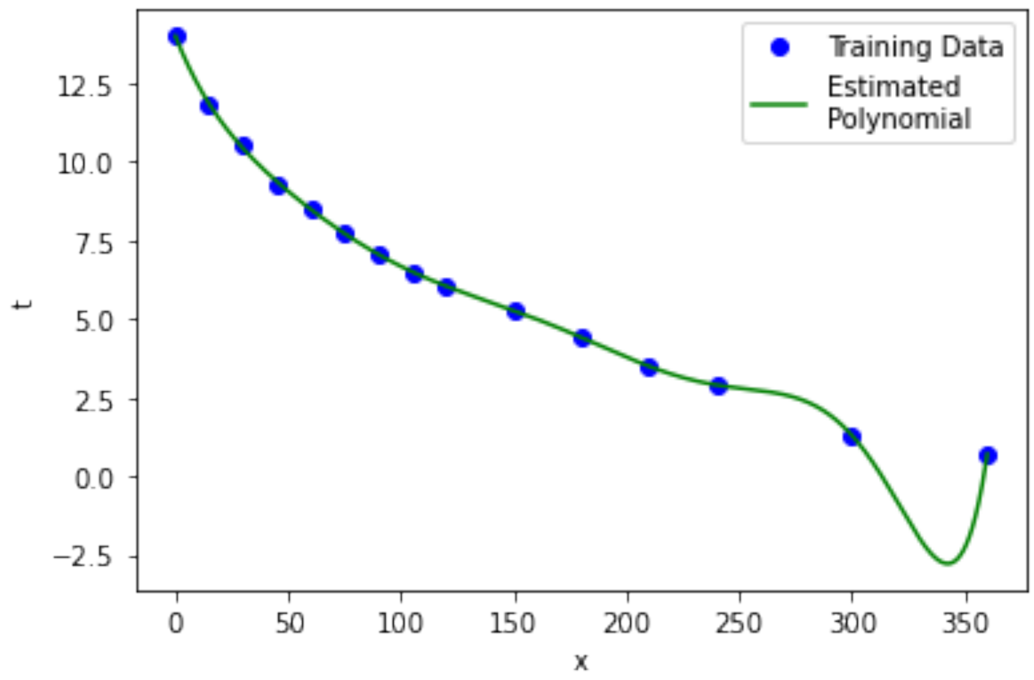
[16]:
```
#Load Data
beerData = np.loadtxt('beer_foam.dat.txt')

plt.scatter(beerData[:,0], beerData[:,1], color = "red")
plt.scatter(beerData[:,0], beerData[:,2], color = "blue")
plt.scatter(beerData[:,0], beerData[:,3], color = "orange")

#Then we can fit the data using the polynomial curve fitting method we derived
x = beerData[:,0]
t = beerData[:,2]
w = fitdata(x,t,M=9)
print(w)
```

```
[ 1.39973990e+01 -1.82312421e-01  3.12156666e-03 -4.21840887e-05
  3.02586205e-07 -5.71862804e-10 -5.81161038e-12  4.05138442e-14
 -9.89564906e-17  8.67513101e-20]
```



[17]:
```
#Now let us use the weights in test
xrange = np.arange(beerData[0,0],beerData[beerData.shape[0]-1,0],0.001)  #get␣
 ↪equally spaced points in the xrange
X = np.array([xrange**m for m in range(w.size)]).T
esty = X@w #compute the predicted value

plotData(x,t,xrange,esty,legend=['Training Data','Estimated\nPolynomial'])

#What will the foam height be at t = ____?
# Initialize 't' as float so below type is correct
t_predict = np.float(450)
x_test = np.array([t_predict**m for m in range(w[:,None].size)]).T
print(x_test)
predicted_height = x_test@w
print(predicted_height)
```

```
[1.00000000e+00 4.50000000e+02 2.02500000e+05 9.11250000e+07
 4.10062500e+10 1.84528125e+13 8.30376562e+15 3.73669453e+18
 1.68151254e+21 7.56680643e+23]
951.5641557009658
```


Training Data
Estimated Polynomial
t
x
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
0
50
100
150
200
250
300
350

02_InClassExercise

September 10, 2020

0.1 Module 02 In Class Exercise

Consider the polynomial curve fitting example discussed in class.

As discussed, when the model order is too small, the training data is generally underfit and when the model order is too high, the result can overfit the training data.

Write a small script of code that mimics our polynomial curve fitting function.

The code should generate simulated data from the true function with added zero-mean Gaussian noise (with the true function assumed to be sinc function).

The code should also generate a separate validation test data set generated in the same way.

Then, after fitting the polynomial to the training data across a range of model orders and evaluated on both the training and testing data, your code should generate a plot similar to the one shown in Figure 1.

Which model order, M, should be used to avoid over-training based on your results?

$$sinc(x) = \begin{cases} 1 & for x = 0 \\ \frac{sin(x)}{x} & otherwise \end{cases} \tag{1}$$

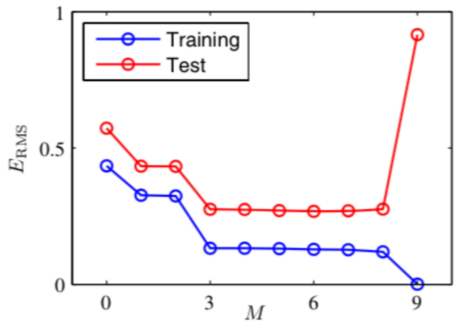


Figure 1: Figure 1.5 from the Bishop textbook. The y-axis corresponds to the root-mean-square error between the predicted and the true value (on either the training data or test data sets). The x-axis corresponds to the model order.

```
[61]: # %load ../HelperCode/plotData.py
      import matplotlib.pyplot as plt
import numpy as np
import math
def plotData(x1,t1,x2,t2,x3=None,t3=None,legend=[]):

    #plot everything
    p1 = plt.plot(x1, t1, 'bo') #plot training data
    p2 = plt.plot(x2, t2, 'g') #plot true value
    if(x3 is not None):
        p3 = plt.plot(x3, t3, 'r')

    #add title, legend and axes labels
    plt.ylabel('t') #label x and y axes
    plt.xlabel('x')

    if(x3 is None):
        plt.legend((p1[0],p2[0]),legend)
    else:
        plt.legend((p1[0],p2[0],p3[0]),legend)



def generateUniformData(N, l, u, gVar):
        '''generateUniformData(N, l, u, gVar): Generate N uniformly spaced data␣
↪points
    in the range [l,u) with zero-mean Gaussian random noise with variance␣
↪gVar'''
        # x = np.random.uniform(l,u,N)
        step = (u-l)/(N);
        x = np.arange(l+step/2,u+step/2,step)
        e = np.random.normal(0,gVar,N)
        t = np.sinc(x) + e
        return x,t
```

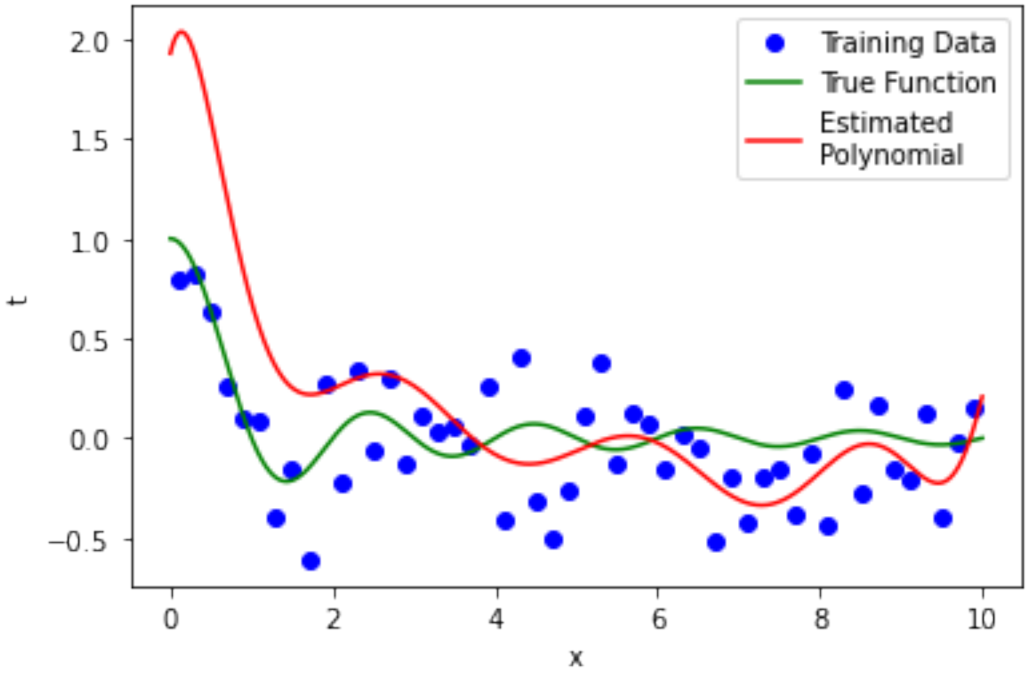
[62]:
```
def fitdata(x,t,M):
    X = np.array([x**m for m in range(M+1)]).T
    w = np.linalg.inv(X.T@X)@X.T@t
    return w
```

[63]:
```
""" ======================  Variable Declaration ========================== """

l = 0 #lower bound on x
u = 10 #upper bound on x
N_train = 50 #number of samples to generate
gVar = .25 #variance of error distribution
M =  12 #regression model order
N_test = 50 #number of test samples to generate
maxM =  13 #Max regression model order
```

[64]:
```
""" ========================  Generate Training Data ======================== """
data_uniform  = np.array(generateUniformData(N_train, l, u, gVar)).T

x1 = data_uniform[:,0]
t1 = data_uniform[:,1]

x2 = np.arange(l,u,0.001)  #get equally spaced points in the xrange
t2 = np.sinc(x2) #compute the true function value
```

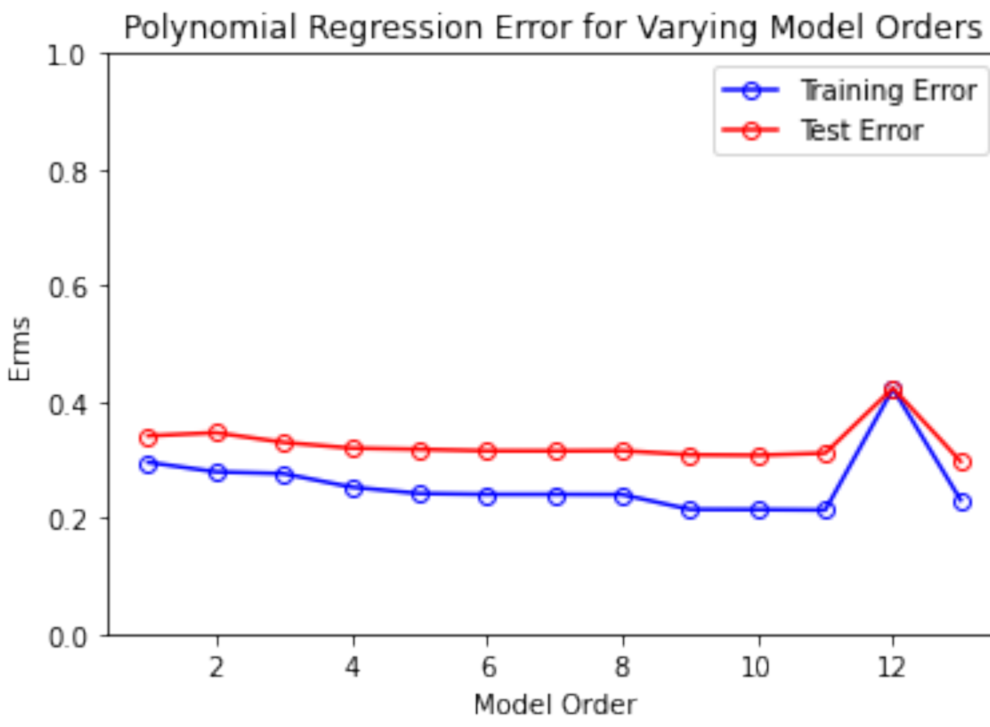
[59]:
```

""" =========================  Train the Model ============================= """
w = fitdata(x1,t1,M)
x3 = np.arange(l,u,0.001)  #get equally spaced points in the xrange
X = np.array([x3**m for m in range(w.size)]).T
t3 = X@w #compute the predicted value

plotData(x1,t1,x2,t2,x3,t3,['Training Data', 'True Function',␣
 ↪'Estimated\nPolynomial'])
print(w)
```

```
[ 1.92983301e+00  1.69019576e+00 -7.50702350e+00  6.84889516e+00
 -2.78506341e+00  5.51415590e-01 -5.05656360e-02  5.25955971e-03
 -2.13244389e-03  4.82407589e-04 -5.27292873e-05  2.83214831e-06
 -6.05228603e-08]
```



```
[60]: """ ======================== Generate Test Data =========================== """

      """This is where you should generate a validation testing data set.  Be sure to␣
       ↪try to generate with different parameters than the training data!   """

      test_data = np.array(generateUniformData(N_test, l, u, gVar)).T
      test_x = test_data[:,0]
      test_t = test_data[:,1]
      train_x = x1
      train_t = t1

      """ ========================  Test the Model ============================== """

      """ This is where you should test the validation set with the trained model """


      """========================================================================"""
      """ ===== Run model fitting and testing over varying model orders  ======== """
      """========================================================================"""
      #initialize error vectors
      erms_train = np.zeros((maxM,1))
      erms_test = np.zeros((maxM,1))

      for M  in range(1,(maxM+1)):
          #fit regression model
          w = fitdata(train_x,train_t,M) #regression weights

          #generate feature vectors
          X_train = np.array([train_x**m for m in range(w.size)]).T #generate feature␣
       ↪vectors
          X_test = np.array([test_x**m for m in range(w.size)]).T #generate feature␣
       ↪vectors

          #estimate labels
          est_train_t = X_train@w #get estimated labels of training data
          est_test_t = X_test@w #get estimated labels of test data

          #calculate instantaneous error
          inst_error_train = (est_train_t - train_t)
          inst_error_test = (est_test_t - test_t)

          #calculate rms error
          erms_train[M-1,0] = math.sqrt((inst_error_train.T@inst_error_train)/
       ↪N_train) #get error for training data
          erms_test[M-1,0] = math.sqrt((inst_error_test.T@inst_error_test)/N_test)␣
       ↪#get error for test data
```

```
#Plot the error vs model order
plt.figure()
p1 = plt.plot(np.arange(1,(maxM+1)),erms_train, marker='o', color='b',␣
↪fillstyle='none')
p2 = plt.plot(np.arange(1,(maxM+1)),erms_test, marker='o', color='r',␣
↪fillstyle='none')
plt.ylim(0,1)
plt.xlabel("Model Order")
plt.ylabel("Erms")
plt.title("Polynomial Regression Error for Varying Model Orders")
plt.legend(("Training Error","Test Error"))
plt.show()
```



```
[ ]:
```

03_EvaluationMetrics

September 15, 2020

1 Error and Evaluation Metrics

- A key step in machine learning algorithm development and testing is determining a good error and evaluation metric.
- Evaluation metrics help us to estimate how well our model is trained and it is important to pick a metric that matches our overall goal for the system.
- Some common evaluation metrics include precision, recall, receiver operating curves, and confusion matrices.

1.0.1 Classification Accuracy and Error

- Classification accuracy is defined as the number of correctly classified samples divided by all samples:

$$\text{accuracy} = \frac{N_{cor}}{N} \tag{1}$$

where N_{cor} is the number of correct classified samples and N is the total number of samples.

- Classification error is defined as the number of incorrectly classified samples divided by all samples:

$$\text{error} = \frac{N_{mis}}{N} \tag{2}$$

where N_{mis} is the number of misclassified samples and N is the total number of samples.

- Suppose there is a 3-class classification problem, in which we would like to classify each training sample (a fish) to one of the three classes (A = salmon or B = sea bass or C = cod).
- Let's assume there are 150 samples, including 50 salmon, 50 sea bass and 50 cod. Suppose our model misclassifies 3 salmon, 2 sea bass and 4 cod.
- Prediction accuracy of our binary classification model is calculated as:

$$\text{accuracy} = \frac{47+48+46}{50+50+50} = \frac{47}{50} \tag{3}$$

- Prediction error is calculated as:

$$\text{error} = \frac{N_{mis}}{N} = \frac{3+2+4}{50+50+50} = \frac{3}{50} \tag{4}$$

1.0.2 Confusion Matrices

- A confusion matrix summarizes the classification accuracy across several classes. It shows the ways in which our classification model is confused when it makes predictions, allowing visualization of the performance of our algorithm. Generally, each row represents the instances of an actual class while each column represents the instances in a predicted class.
- If our classifier is trained to distinguish between salmon, sea bass and cod, then we can summarize the prediction result in the confusion matrix as follows:

Actual/Predicted	Salmon	Sea bass	Cod
Salmon	47	2	1
Sea Bass	2	48	0
Cod	0	0	50

- In this confusion matrix, of the 50 actual salmon, the classifier predicted that 2 are sea bass, 1 is cod incorrectly and 47 are labeled salmon correctly. All correct predictions are located in the diagonal of the table. So it is easy to visually inspect the table for prediction errors, as they will be represented by values outside the diagonal.

1.0.3 TP, FP, TN, and FN

- True positive (TP): correctly predicting event values
- False positive (FP): incorrectly calling non-events as an event
- True negative (TN): correctly predicting non-event values
- False negative (FN): incorrectly labeling events as non-event
- Precision is also called positive predictive value.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5}$$

- Recall is also called true positive rate, probability of detection

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6}$$

- Fall-out is also called false positive rate, probability of false alarm.

$$\text{Fall-out} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{7}$$

- *Consider the salmon/non-salmon classification problem, what are the TP, FP, TN, FN values?*

Actual/Predicted	Salmon	Non-Salmon
Salmon	47	3
Non-Salmon	2	98

1.0.4 ROC curves

- The Receiver Operating Characteristic (ROC) curve is a plot between the true positive rate (TPR) and the false positive rate (FPR), where the TPR is defined on the y-axis and FPR is defined on the x-axis.
- $TPR = TP/(TP + FN)$ is defined as ratio between true positive prediction and all real positive samples. The definition used for FPR in a ROC curve is often problem dependent. For example, for detection of targets in an area, FPR may be defined as the ratio between the number of false alarms per unit area (FA/m^2). In another example, if you have a set number of images and you are looking for targets in these collection of images, FPR may be defined as the number of false alarms per image. In some cases, it may make the most sense to simply use the Fall-out or false positive rate.
- Given a binary classifier and its threshold, the (x,y) coordinates of ROC space can be calculated from all the prediction result. You trace out a ROC curve by varying the threshold to get all of the points on the ROC.
- The diagonal between (0,0) and (1,1) separates the ROC space into two areas, which are left up area and right bottom area. The points above the diagonal represent good classification (better than random guess) which below the diagonal represent bad classification (worse than random guess).
- *What is the perfect prediction point in a ROC curve?*

1.0.5 Precision-Recall curves

- ROC curves trace out the TPR vs. FPR over many thresholds.
- Similarly, other metrics can be plotted over many thresholds. Another common example is Precision-Recall curves
- PR curves are generated the same way as ROC curves, however, instead of plooting TPR vs. FPR, Precision vs. Recall (as defined above) are plotted over many thresholds.
- *What does the perfect PR curve look like?*
- PR curves are often preferred over ROC curves in cases of severely imbalanced data.
- Similar to ROC and PR curves, any statistics that can be computed via the confusion matrix, can be plotted over all possible thresholds.

1.0.6 MSE and MAE

- *Mean Square Error* (MSE) is the average of the squared error between prediction and actual observation.
- For each sample $\mathbf{x}_i$, the prediction value is y_i and the actual output is d_i. The MSE is

$$MSE = \sum_{i=1}^{n} \frac{(d_i - y_i)^2}{n} \tag{8}$$

- *Root Mean Square Error* (RMSE) is simply the square root the MSE.

$$RMSE = \sqrt{MSE} \tag{9}$$

- *Mean Absolute Error* (MAE) is the average of the absolute error.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|d_i - y_i| \tag{10}$$

1.0.7 Final Thoughts/Summary

Of course, there are many other evaluation metrics. These are just a few of the most commonly used. In practice, the *best* evaluation metrics are those that provide you insight into your problem - particularly when it is not tied closely to your objective function.

It is important to remember Goodhart's Law when using evaluation metrics (https://en.wikipedia.org/wiki/Goodhart%27s_law):

"When a measure becomes a target, it ceases to be a good measure."

03_ExpDesign

September 15, 2020

1 Experimental Design in Machine Learning

- How can we estimated the expected performance of a machine learning algorithm for a particular application?
- How do we select between machine learning algorithms (or parameter settings)?
- Commonly, we split a data set into three groups: training, validation and test. (Only for evaluation purposes - often use ALL data for a final training of the system)
- The training and validation sets may be rotated (e.g., using cross-validation)
- As previously discussed, we cannot rely on training (or validation) performance (because of the potential for over-fitting) to help us answer the questions above
- Our performance estimate is only as good as our test set is representative of the true test data in application
- We also cannot rely on one training run of the algorithm:
  - variations in training/validation sets
  - random factors during training (e.g., random initialization, local optima, etc.)
- The No Free Lunch Theorem: There is no universally best algorithm. For any machine learning algorithm there are sets of data where it works well and sets where it works poorly
- Performance of an algorithm can be determined using any of the measures we discussed previously (e.g., error rate, accuracy, ROC curves, Perf-Recall curves, etc) but also in terms of:
  - Risk
  - Running time
  - Training time
  - Run time storage/memory
  - Train time storage/memory
  - Computational complexity
  - Interpretability
  - etc…
- The output of a trained learning system depends on:
  - Controllable parameters: hyperparameters/settings of the algorithm/algorithm design choices
  - Uncontrollable parameters: noise in data, any randomness in training
- To fully test a system, you want to try to evaluate each of these parameters separately. However, this is often not easily done.

- Various strategies:
  - Best guess
  - Varying one factor at a time
  - Full/Partial Factorial design
- If there is randomization in your experiment, need to run multiple times (replicate experiments) to estimate the expected result (e.g., mean and variance of performance)
- If there are real-world impacts to your runs (e.g., machine warming up, time of day, etc), need to randomize your trials across factors
- Need to ensure we are comparing the parameters and algorithms we are interested and not any confounding factors (e.g., if you want to compare one parameter - ensure everything else is fixed)

[ ]:
```
* When conducting experiments:
    - Understand the goal of your study
    - Determine your evaluation measure(s)
    - Determine what factors to vary and how to vary them
    - Design your experiment (and get estimate of how long it will take using a␣
↪couple trial subset runs)
    - Perform th experiment
    - Analy
```

04_MaximumLikelihood

September 15, 2020

1 Maximum Likelihood

- *Method of Maximum Likelihood:*
  - A *data likelihood* is how likely the data is given the parameter set
  - So, if we want to maximize how likely the data is to have come from the model we fit, we should find the parameters that maximize the likelihood
  - A common trick to maximizing the likelihood is to maximize the log likelihood. Often makes the math much easier. *Why can we maximize the log likelihood instead of the likelihood and still get the same answer?*
  - Consider: $\max \ln \exp \left\{ -\frac{1}{2} \left(y(x_n, \mathbf{w}) - t_n \right)^2 \right\}$ We go back to our original objective.

1.1 Maximum Likelihood for the Bernoulli Distribution

- Lets look at this in terms of binary variables, e.g., Flipping a coin: $X = 1$ is heads, $X = 0$ is tails
- Let μ be the probability of heads. If we know μ, then: $P(x = 1|\mu) = \mu$ and $P(x = 0|\mu) = 1 - \mu$

$$P(x|\mu) = \mu^x (1-\mu)^{1-x} = \begin{cases} \mu & \text{if } x = 1 \\ 1-\mu & \text{if } x = 0 \end{cases} \tag{1}$$

- This is called the *Bernoulli* distribution. The mean and variance of a Bernoulli distribution is:

$$E[x] = \mu \tag{2}$$

$$E\left[(x-\mu)^2\right] = \mu(1-\mu) \tag{3}$$

- So, suppose we conducted many Bernoulli trials (e.g., coin flips) and we want to estimate μ

1.1.1 Method: Maximum Likelihood

$$p(\mathscr{D}|\mu) = \prod_{n=1}^{N} p(x_n|\mu) \tag{4}$$

$$= \prod_{n=1}^{N} \mu^{x_n} (1-\mu)^{1-x_n} \tag{5}$$

- Maximize : (*What trick should we use?*)

$$\mathscr{L} = \sum_{n=1}^{N} x_n \ln \mu + (1 - x_n) \ln(1-\mu) \tag{6}$$

$$
\begin{aligned}
\frac{\partial \mathscr{L}}{\partial \mu} = 0 &= \frac{1}{\mu}\sum_{n=1}^{N} x_n - \frac{1}{1-\mu}\sum_{n=1}^{N}(1-x_n) && (7)\\
0 &= \frac{(1-\mu)\sum_{n=1}^{N} x_n - \mu \sum_{n=1}^{N}(1-x_n)}{\mu(1-\mu)} && (8)\\
0 &= \sum_{n=1}^{N} x_n - \mu\sum_{n=1}^{N} x_n - \mu\sum_{n=1}^{N} 1 + \mu\sum_{n=1}^{N} x_n && (9)\\
0 &= \sum_{n=1}^{N} x_n - \mu N && (10)\\
\mu &= \frac{1}{N}\sum_{n=1}^{N} x_n = \frac{m}{N} && (11)
\end{aligned}
$$

where m is the number of successful trials.

- So, if we flip a coin 1 time and get heads, then $\mu = 1$ and probability of getting tails is 0. *Would you believe that? We need a prior!*

1.2 The Gaussian Distribution:

- Consider a univariate Gaussian distribution:

$$
\mathscr{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \qquad (12)
$$

- σ^2 is the variance OR $\frac{1}{\sigma^2}$ is the *precision*
- So, as λ gets big, variance gets smaller/tighter. As λ gets small, variance gets larger/wider.
- The Gaussian distribution is also called the *Normal* distribution.
- We will often write $N(x|\mu, \sigma^2)$ to refer to a Gaussian with mean μ and variance σ^2.
- *What is the multi-variate Gaussian distribution?*
- What is the expected value of x for the Gaussian distribution?

$$
\begin{aligned}
E[x] &= \int x p(x) dx && (13)\\
&= \int x \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} dx && (14)
\end{aligned}
$$

- *Change of variables:* Let

$$
\begin{aligned}
y &= \frac{x-\mu}{\sigma} \rightarrow x = \sigma y + \mu && (15)\\
dy &= \frac{1}{\sigma} dx \rightarrow dx = \sigma dy && (16)
\end{aligned}
$$

- Plugging this into the expectation:

$$\begin{aligned} E[x] &= \int (\sigma y + \mu) \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2}y^2\right\} \sigma dy && (17) \\ &= \int \frac{\sigma y}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}y^2\right\} dy + \int \frac{\mu}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}y^2\right\} dy && (18) \end{aligned}$$

- The first term is an odd function: $f(-y) = -f(y)$ So, $E[x] = 0 + \mu = \mu$

1.3 MLE of Mean of Gaussian

- Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be samples from a multi-variance Normal distribution with known covariance matrix and an unknown mean. Given this data, obtain the ML estimate of the mean vector.

$$p(\mathbf{x}_k|\mu) = \frac{1}{(2\pi)^{\frac{l}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}_k - \mu)^T \Sigma^{-1} (\mathbf{x}_k - \mu)\right) \quad (19)$$

- We can define our likelihood given the N data points. We are assuming these data points are drawn independently but from an identical distribution (i.i.d.):

$$\prod_{n=1}^{N} p(\mathbf{x}_n|\mu) = \prod_{n=1}^{N} \frac{1}{(2\pi)^{\frac{l}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T \Sigma^{-1} (\mathbf{x}_n - \mu)\right) \quad (20)$$

- We can apply our "trick" to simplify

$$\begin{aligned} \mathscr{L} &= \ln \prod_{n=1}^{N} p(\mathbf{x}_n|\mu) = \ln \prod_{n=1}^{N} \frac{1}{(2\pi)^{\frac{l}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T \Sigma^{-1} (\mathbf{x}_n - \mu)\right) && (21) \\ &= \sum_{n=1}^{N} \ln \frac{1}{(2\pi)^{\frac{l}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T \Sigma^{-1} (\mathbf{x}_n - \mu)\right) && (22) \\ &= \sum_{n=1}^{N} \left(\ln \frac{1}{(2\pi)^{\frac{l}{2}} |\Sigma|^{\frac{1}{2}}} + \left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T \Sigma^{-1} (\mathbf{x}_n - \mu)\right)\right) && (23) \\ &= -N \ln(2\pi)^{\frac{l}{2}} |\Sigma|^{\frac{1}{2}} + \sum_{n=1}^{N} \left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T \Sigma^{-1} (\mathbf{x}_n - \mu)\right) && (24) \end{aligned}$$

- Now, lets maximize:

$$\frac{\partial \mathscr{L}}{\partial \mu} = \frac{\partial}{\partial \mu}\left[-N\ln(2\pi)^{\frac{l}{2}}\left|\Sigma\right|^{\frac{1}{2}} + \sum_{n=1}^{N}\left(-\frac{1}{2}(\mathbf{x}_n-\mu)^T\Sigma^{-1}(\mathbf{x}_n-\mu)\right)\right] = 0 \tag{25}$$

$$\rightarrow \sum_{n=1}^{N}\Sigma^{-1}(\mathbf{x}_n-\mu) = 0 \tag{26}$$

$$\rightarrow \sum_{n=1}^{N}\Sigma^{-1}\mathbf{x}_n = \sum_{n=1}^{N}\Sigma^{-1}\mu \tag{27}$$

$$\rightarrow \Sigma^{-1}\sum_{n=1}^{N}\mathbf{x}_n = \Sigma^{-1}\mu N \tag{28}$$

$$\rightarrow \sum_{n=1}^{N}\mathbf{x}_n = \mu N \tag{29}$$

$$\rightarrow \frac{\sum_{n=1}^{N}\mathbf{x}_n}{N} = \mu \tag{30}$$

$$\tag{31}$$

- So, the ML estimate of μ is the sample mean!

Maximum Likelihood and Maximum A Posterior

- We looked at the regularization term as a *penalty* term in the objective function. There is another way to interpret the regularization term as well. Specifically, there is a *Bayesian* interpretation.

$$\begin{aligned}
\min E^*(\mathbf{w}) &= \max -E^*(\mathbf{w}) \\
&= \max \exp\left\{-E^*(\mathbf{w})\right\} \\
&= \max \exp\left\{-\frac{1}{2}\sum_{n=1}^{N}(y(x_n,\mathbf{w})-t_n)^2 - \frac{\lambda}{2}\|\mathbf{w}\|_2^2\right\} \\
&= \max \exp\left\{-\frac{1}{2}\sum_{n=1}^{N}(y(x_n,\mathbf{w})-t_n)^2\right\}\exp\left\{-\frac{1}{2}\lambda\|\mathbf{w}\|_2^2\right\} \\
&= \max \prod_{n=1}^{N}\exp\left\{-\frac{1}{2}(y(x_n,\mathbf{w})-t_n)^2\right\}\exp\left\{-\frac{1}{2}\lambda\|\mathbf{w}\|_2^2\right\}
\end{aligned}$$

- So, this is a maximization of the *data likelihood* with a *prior*: $p(\mathbf{X}|\mathbf{w})p(\mathbf{w})$
- *Method of Maximum Likelihood:*
  - A *data likelihood* is how likely the data is given the parameter set
  - So, if we want to maximize how likely the data is to have come from the model we fit, we should find the parameters that maximize the likelihood
  - A common trick to maximizing the likelihood is to maximize the log likelihood. Often makes the math much easier. *Why can we maximize the log likelihood instead of the likelihood and still get the same answer?*
  - Consider: $\max \ln \exp\left\{-\frac{1}{2}(y(x_n,\mathbf{w})-t_n)^2\right\}$ We go back to our original objective.

- *Method of Maximum A Posteriori (MAP):*
  - Bayes Rule: $p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$
  - Consider: $p(\mathbf{w}|\mathscr{D}) = \frac{p(\mathscr{D}|\mathbf{w})p(\mathbf{w})}{p(\mathscr{D})}$, i.e., posterior $\propto$ likelihood $\times$ prior

Maximum Likelihood vs. Maximum A Posteriori (MAP)

- Lets look at this in terms of binary variables, e.g., Flipping a coin: $X = 1$ is heads, $X = 0$ is tails
- Let μ be the probability of heads. If we know μ, then: $P(x=1|\mu)=\mu$ and $P(x=0|\mu)=1-\mu$

$$P(x|\mu) = \mu^x(1-\mu)^{1-x} = \begin{cases} \mu & \text{if } x = 1 \\ 1-\mu & \text{if } x = 0 \end{cases}$$

- This is called the *Bernoulli* distribution. The mean and variance of a Bernoulli distribution is:

$$E[x] = \mu$$
$$E\left[(x-\mu)^2\right] = \mu(1-\mu)$$

- So, suppose we conducted many Bernoulli trials (e.g., coin flips) and we want to estimate μ

Method: Maximum Likelihood

$$\begin{aligned}
p(\mathscr{D}|\mu) &= \prod_{n=1}^{N} p(x_n|\mu) \\
&= \prod_{n=1}^{N} \mu^{x_n}(1-\mu)^{1-x_n}
\end{aligned}$$

- Maximize : (*What trick should we use?*)

$$\mathscr{L} = \sum_{n=1}^{N} x_n \ln\mu + (1-x_n)\ln(1-\mu)$$

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial \mu} = 0 &= \frac{1}{\mu}\sum_{n=1}^{N} x_n - \frac{1}{1-\mu}\sum_{n=1}^{N}(1-x_n)\\ 0 &= \frac{(1-\mu)\sum_{n=1}^{N} x_n - \mu\sum_{n=1}^{N}(1-x_n)}{\mu(1-\mu)}\\ 0 &= \sum_{n=1}^{N} x_n - \mu\sum_{n=1}^{N} x_n - \mu\sum_{n=1}^{N} 1 + \mu\sum_{n=1}^{N} x_n\\ 0 &= \sum_{n=1}^{N} x_n - \mu N\\ \mu &= \frac{1}{N}\sum_{n=1}^{N} x_n = \frac{m}{N}\end{aligned}$$

where m is the number of successful trials.

- So, if we flip a coin 1 time and get heads, then $\mu = 1$ and probability of getting tails is 0. *Would you believe that? We need a prior!*

Method: Maximum A Posteriori:

- Look at several independent trials. Consider N = 3 and m = 2 (N is number of trials, m is number of successes) and look at all ways to get 2 H and 1 T:
  - H H T $\rightarrow \mu\mu(1-\mu) = \mu^2(1-\mu)$
  - H T H $\rightarrow \mu(1-\mu)\mu = \mu^2(1-\mu)$
  - T H H $\rightarrow (1-\mu)\mu\mu = \mu^2(1-\mu)$
- $\begin{pmatrix} 3 \\ 2 \end{pmatrix} \mu^2(1-\mu) \rightarrow \begin{pmatrix} N \\ m \end{pmatrix} \mu^m(1-\mu)^{N-m} = \frac{N!}{(N-m)!m!}\mu^m(1-\mu)^{N-m}$
- This is the Binomial Distribution, gives the probability of m observations of $x = 1$ out of N independent trails
- So, what we saw is that we need a prior. We want to incorporate our prior belief. Let us place a prior on μ

$$\begin{aligned}Beta(\mu|a,b) &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\mu^{a-1}(1-\mu)^{b-1}\\ E[\mu] &= \frac{a}{a+b}\\ Var[\mu] &= \frac{ab}{(a+b)^2(a+b+1)}\end{aligned}$$

- Note: $\Gamma(x) = \int_0^\infty u^{x-1}e^{-u}\,du$ and when x is an integer, then it simplifys to $(x-1)!$
- Calculation of the posterior, Take $N = m + l$ observations:

$$\begin{aligned}p(\mu|m,l,a,b) &\propto Bin(m,l|\mu)Beta(\mu|a,b)\\ &\propto \mu^m(1-\mu)^l\mu^{a-1}(1-\mu)^{b-1}\\ &= \mu^{m+a-1}(1-\mu)^{l+b-1}\end{aligned}$$

- What does this look like? Beta: $a \leftarrow m + a$, $b \leftarrow l + b$
- So, what's the posterior?

$$p(\mu|m,l,a,b) = \frac{\Gamma(m+a+l+b)}{\Gamma(m+a)\Gamma(l+b)}\mu^{m+a-1}(1-\mu)^{l+b-1}$$

- *Conjugate Prior Relationship:* When the posterior is the same form as the prior
- Now we can maximize the (log of the) posterior:

$$\begin{aligned}&\max_\mu((m+a-1)\ln\mu + (l+b-1)\ln(1-\mu))\\ \frac{\partial \mathscr{L}}{\partial \mu} = 0 &= \frac{m+a-1}{\mu} - \frac{l+b-1}{1-\mu}\\ &= (1-\mu)(m+a-1) - \mu(l+b-1)\\ &= (m+a-1) - \mu(m+a-1) - \mu(l+b-1)\\ \mu &= \frac{m+a-1}{m+a+l+b-2}\end{aligned}$$

- This is the MAP solution. *So, what happens now when you flip one heads, two heads, etc.?*
- Discuss online updating of the prior. Eventually the data takes over the prior.

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        import math
        %matplotlib inline

        priorA = 2
        priorB = 2
        def plotBeta(a=priorA,b=priorB):
            '''plotBeta(a=1,b=1): Plot plot beta distribution with parameters a and b'''
            xrange = np.arange(0,1,0.001)  #get equally spaced points in the xrange
            normconst = math.gamma(a+b)/(math.gamma(a)*math.gamma(b))
            beta = normconst*xrange**(a-1)*(1-xrange)**(b-1)
            fig = plt.figure()
            p1 = plt.plot(xrange,beta, 'g')
            plt.show()


        #Beta Distribution
        # plotBeta(priorA,priorB);

        trueMu = 0.5
        numFlips = 100
        flipResult = []
        for flip in range(numFlips):
            flipResult.append(np.random.binomial(1,trueMu,1)[0])
            print(flipResult)
            print('Frequentist/Maximum Likelihood Probability of Heads:' + str(sum(flipResult)/len(flipRe
            print('Bayesian/MAP Probability of Heads:' + str((sum(flipResult)+priorA-1)/(len(flipResult)+
            if (input("Hit enter to continue, or q to quit...\n") == "q"):
                print("quitting...\n")
                break
```

The Gaussian Distribution:

- Consider a univariate Gaussian distribution:
$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\}$$
- σ^2 is the variance OR $\frac{1}{\sigma^2}$ is the *precision*
- So, as λ gets big, variance gets smaller/tighter. As λ gets small, variance gets larger/wider.
- The Gaussian distribution is also called the *Normal* distribution.
- We will often write $N(x|\mu,\sigma^2)$ to refer to a Gaussian with mean μ and variance σ^2.
- *What is the multi-variate Gaussian distribution?*
- What is the expected value of x for the Gaussian distribution?
$$\begin{aligned} E[x] &= \int xp(x)dx \\ &= \int x\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\}dx \end{aligned}$$
- *Change of variables:* Let
$$\begin{aligned} y &= \frac{x-\mu}{\sigma} \rightarrow x = \sigma y + \mu \\ dy &= \frac{1}{\sigma}dx \rightarrow dx = \sigma dy \end{aligned}$$
- Plugging this into the expectation:
$$\begin{aligned} E[x] &= \int(\sigma y+\mu)\frac{1}{\sqrt{2\pi\sigma}}\exp\left\{-\frac{1}{2}y^2\right\}\sigma dy \\ &= \int\frac{\sigma y}{\sqrt{2\pi}}\exp\left\{-\frac{1}{2}y^2\right\}dy + \int\frac{\mu}{\sqrt{2\pi}}\exp\left\{-\frac{1}{2}y^2\right\}dy \end{aligned}$$
- The first term is an odd function: $f(-y) = -f(y)$ So, $E[x] = 0 + \mu = \mu$

MLE of Mean of Gaussian

- Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be samples from a multi-variance Normal distribution with known covariance matrix and an unknown mean. Given this data, obtain the ML estimate of the mean vector.

$$p(\mathbf{x}_k|\mu) = \frac{1}{(2\pi)^{\frac{l}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{x}_k - \mu)^T\Sigma^{-1}(\mathbf{x}_k - \mu)\right)$$

- We can define our likelihood given the N data points. We are assuming these data points are drawn independently but from an identical distribution (i.i.d.):

$$\prod_{n=1}^{N} p(\mathbf{x}_n|\mu) = \prod_{n=1}^{N}\frac{1}{(2\pi)^{\frac{l}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T\Sigma^{-1}(\mathbf{x}_n - \mu)\right)$$

- We can apply our "trick" to simplify

$$\begin{aligned}
\mathscr{L} &= \ln\prod_{n=1}^{N} p(\mathbf{x}_n|\mu) = \ln\prod_{n=1}^{N}\frac{1}{(2\pi)^{\frac{l}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T\Sigma^{-1}(\mathbf{x}_n - \mu)\right)\\
&= \sum_{n=1}^{N}\ln\frac{1}{(2\pi)^{\frac{l}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T\Sigma^{-1}(\mathbf{x}_n - \mu)\right)\\
&= \sum_{n=1}^{N}\left(\ln\frac{1}{(2\pi)^{\frac{l}{2}}|\Sigma|^{\frac{1}{2}}} + \left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T\Sigma^{-1}(\mathbf{x}_n - \mu)\right)\right)\\
&= -N\ln(2\pi)^{\frac{l}{2}}|\Sigma|^{\frac{1}{2}} + \sum_{n=1}^{N}\left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T\Sigma^{-1}(\mathbf{x}_n - \mu)\right)
\end{aligned}$$

- Now, lets maximize:

$$\begin{aligned}
\frac{\partial\mathscr{L}}{\partial\mu} &= \frac{\partial}{\partial\mu}\left[-N\ln(2\pi)^{\frac{l}{2}}|\Sigma|^{\frac{1}{2}} + \sum_{n=1}^{N}\left(-\frac{1}{2}(\mathbf{x}_n - \mu)^T\Sigma^{-1}(\mathbf{x}_n - \mu)\right)\right] = 0\\
&\rightarrow \sum_{n=1}^{N}\Sigma^{-1}(\mathbf{x}_n - \mu) = 0\\
&\rightarrow \sum_{n=1}^{N}\Sigma^{-1}\mathbf{x}_n = \sum_{n=1}^{N}\Sigma^{-1}\mu\\
&\rightarrow \Sigma^{-1}\sum_{n=1}^{N}\mathbf{x}_n = \Sigma^{-1}\mu N\\
&\rightarrow \sum_{n=1}^{N}\mathbf{x}_n = \mu N\\
&\rightarrow \frac{\sum_{n=1}^{N}\mathbf{x}_n}{N} = \mu
\end{aligned}$$

- So, the ML estimate of μ is the sample mean!

MAP of Mean of Gaussian

- To get a MAP estimate of the mean of a Gaussian, we apply a prior distribution and maximize the posterior.
- Lets use a Gaussian prior on the mean (because it has a *conjugate prior* relationship)

$$
\begin{aligned}
p(\mu|X, \mu_0, \sigma_0^2, \sigma^2) &\propto \mathcal{N}(X|\mu, \sigma^2)\mathcal{N}(\mu|\mu_0, \sigma_0^2) \\
&= \prod_{n=1}^{N}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2\sigma^2}(x_n-\mu)^2\right\}\right)\frac{1}{\sqrt{2\pi\sigma_0^2}}\exp\left\{-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right\} \\
\mathcal{L} &= -\frac{N}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 - \frac{N}{2}\ln(2\pi\sigma_0^2) - \frac{N}{2\sigma_0^2}(\mu-\mu_0)^2 \\
\frac{\partial\mathcal{L}}{\partial\mu} &= -\frac{N}{\sigma^2}\mu - \frac{N}{\sigma_0^2}\mu + \frac{N}{\sigma_0^2}\mu_0 + \frac{1}{\sigma^2}\sum_{n=1}^{N}x_n = 0 \\
N\mu\left(\frac{\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}\right) &= \frac{1}{\sigma^2}\sum_{n=1}^{N}x_n + \frac{1}{\sigma_0^2}\mu_0 \\
\mu_{MAP} &= \frac{\sigma_0^2}{N\sigma_0^2+N\sigma^2}\sum_{n=1}^{N}x_n + \frac{\mu_0\sigma^2}{N\sigma_0^2+N\sigma^2}
\end{aligned}
$$

- *Does this result make sense?*

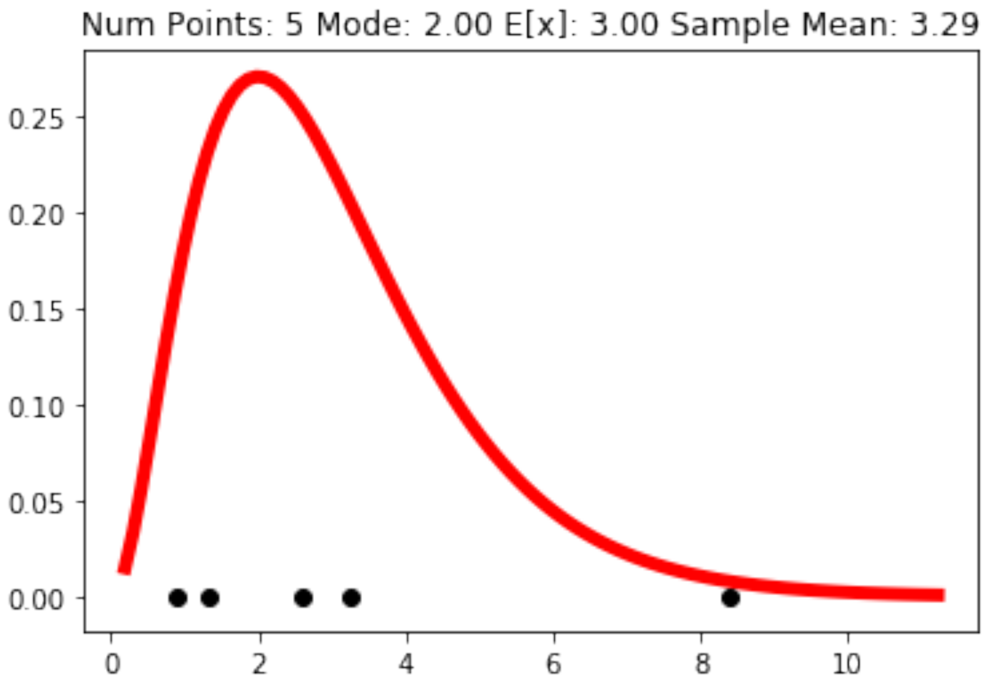
```
In [ ]:
```

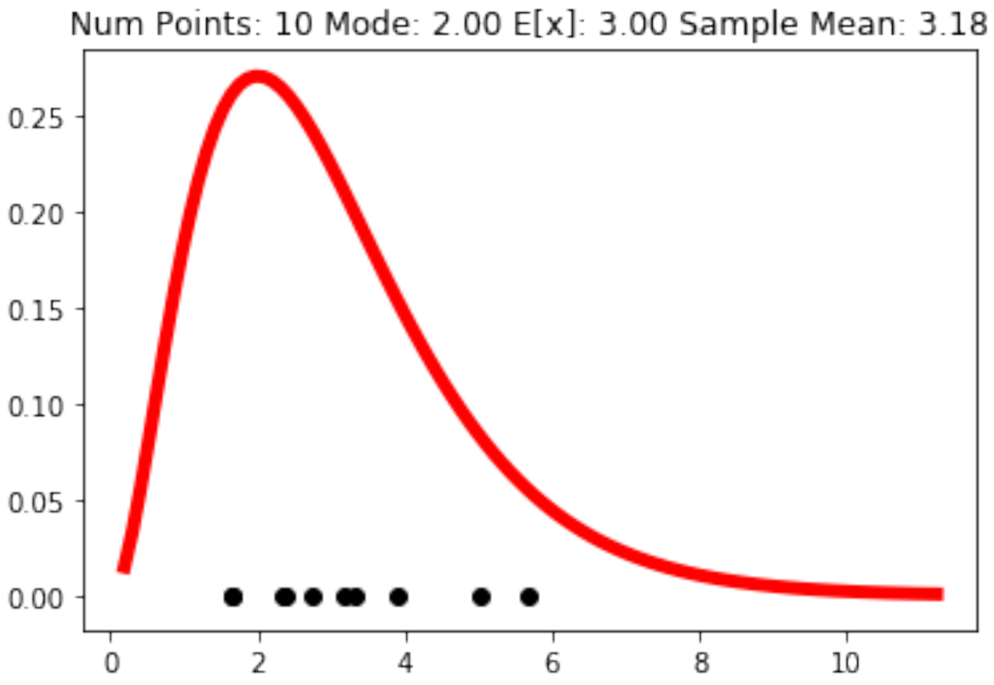
05_B_Conjugate Priors and Bayesian Linear Regression

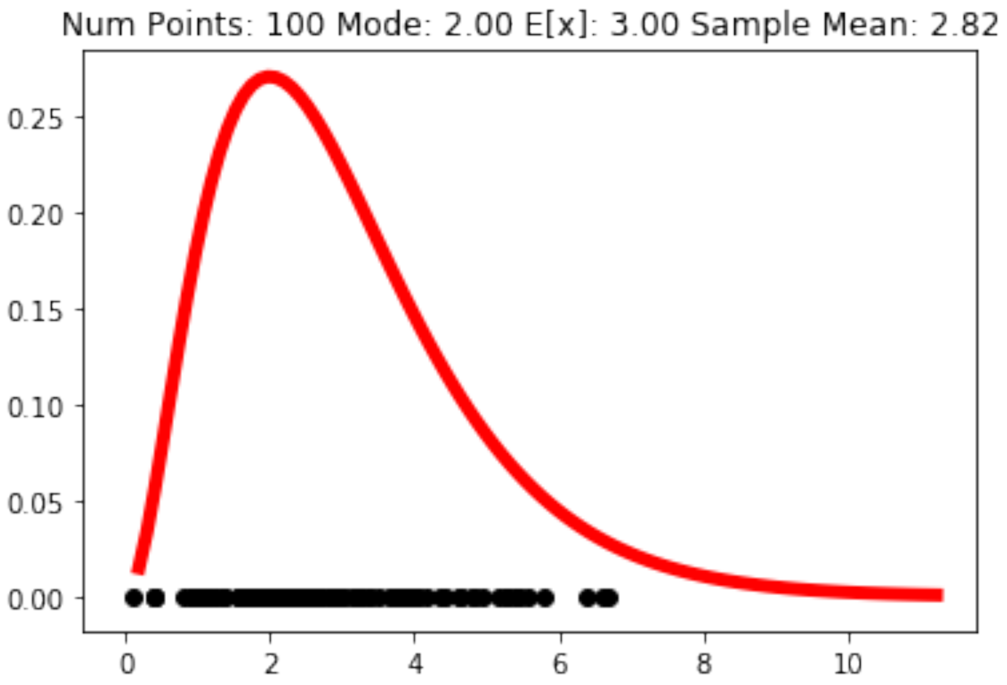
September 15, 2020

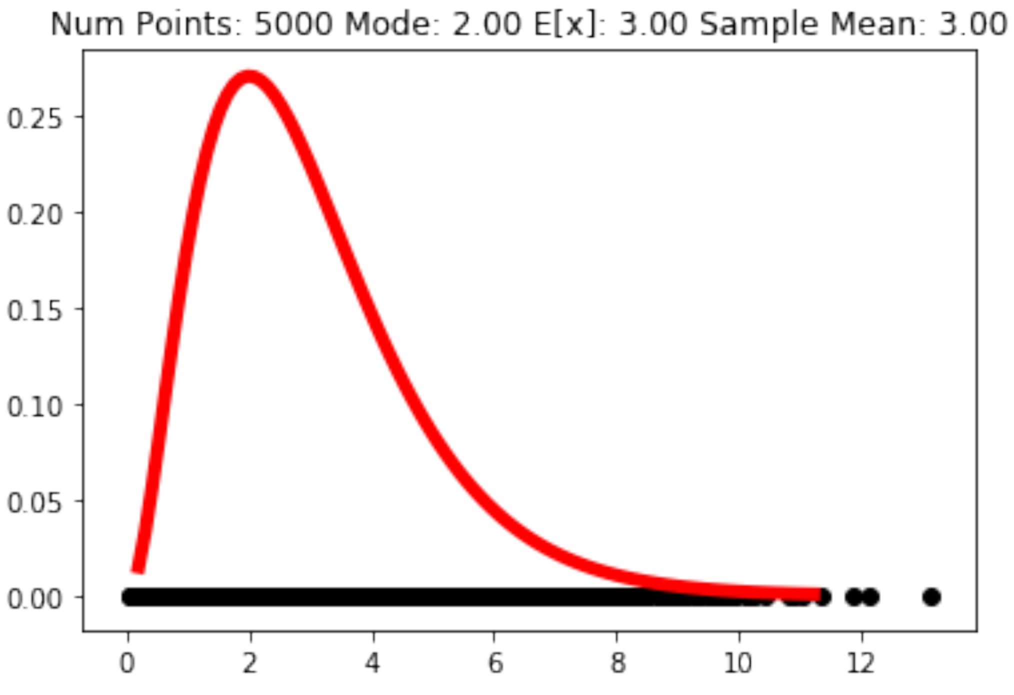
1 Sample Mean vs. Mode vs. Expected Value

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gamma
from scipy.stats import multivariate_normal
import textwrap
import math
%matplotlib inline

def sampleMeanEx():
        '''sampleMeanEx()'''
        nSamples = (5, 10, 100, 5000)
        a = 3
        b = 1
        for i in range(len(nSamples)):
                fig = plt.figure()
                ax = fig.add_subplot(*[1,1,1])
                draws = np.random.gamma(shape=a,scale=b,size=nSamples[i])
                mode = (a-1)*b
                expectedv = a*b
                samplemean = sum(draws)/len(draws)
                x = np.linspace(gamma.ppf(0.001, a, scale=b), gamma.ppf(0.999,␣
 ↪a, scale=b), 100)
                ax.plot(x, gamma.pdf(x, a, scale=b), 'r-', lw=5)
                ax.scatter(draws, np.zeros(len(draws)), c='k')
                myTitle = 'Num Points: ' + str(nSamples[i]) + ' Mode: ' +␣
 ↪str("%.2f"%mode) + ' E[x]: ' + str("%.2f"%expectedv) + ' Sample Mean: ' +␣
 ↪str("%.2f"%samplemean)
                ax.set_title("\n".join(textwrap.wrap(myTitle, 100)))
                plt.show()

sampleMeanEx()
```


Num Points: 5 Mode: 2.00 E[x]: 3.00 Sample Mean: 3.29
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10


Num Points: 10 Mode: 2.00 E[x]: 3.00 Sample Mean: 3.18
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10


Num Points: 100 Mode: 2.00 E[x]: 3.00 Sample Mean: 2.82
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10


Num Points: 5000 Mode: 2.00 E[x]: 3.00 Sample Mean: 3.00
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10
12

2 Conjugate Priors

- Last class we mentioned the concept of *conjugate priors*
- Two distributions have a conjugate prior relationship when the form of the posterior is the same as the form of the prior.
- For example, a Gaussian distribution is a conjugate prior for the mean of a Gaussian as shown in the following:

$$
\begin{aligned}
p(\mu|\mathbf{X}) &\propto p(\mathbf{X}|\mu)p(\mu) && (1)\\
&= \prod_{i=1}^{N} \mathcal{N}(x_i|\mu,\sigma^2)\mathcal{N}(\mu|\mu_0,\sigma_0^2) && (2)\\
&= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\frac{(x_i-\mu)^2}{\sigma^2}\right\} \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{1}{2}\frac{(\mu-\mu_0)^2}{\sigma_0^2}\right\} && (3)\\
&= \frac{1}{\sqrt{2\pi\sigma^2}}\frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left\{\sum_{i=1}^{N}\left(-\frac{1}{2}\frac{(x_i-\mu)^2}{\sigma^2}\right)-\frac{1}{2}\frac{(\mu-\mu_0)^2}{\sigma_0^2}\right\} && (4)\\
&= \frac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{1}{2}\left(\sum_{i=1}^{N}\frac{(x_i-\mu)^2}{\sigma^2}+\frac{(\mu-\mu_0)^2}{\sigma_0^2}\right)\right\} && (5)\\
&= \frac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{1}{2}\left(\frac{\sum_{i=1}^{N}x_i^2-2\sum_{i=1}^{N}x_i\mu+\mu^2N}{\sigma^2}+\frac{\mu^2-2\mu\mu_0+\mu_0^2}{\sigma_0^2}\right)\right\} && (6)\\
&= \frac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{1}{2}\left(\mu^2\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)-2\mu\left(\frac{\sum_{i=1}^{N}x_i}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right)+\frac{\sum_{i=1}^{N}x_i^2}{\sigma^2}+\frac{\mu_0^2}{\sigma_0^2}\right)\right\} && (7)\\
&= \frac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{1}{2}\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\left(\mu^2-2\mu\left(\frac{\sum_{i=1}^{N}x_i}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right)\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)^{-1}\right)\right\} && (8)\\
&\quad \exp\left\{\frac{\sum_{i=1}^{N}x_i^2}{\sigma^2}+\frac{\mu_0^2}{\sigma_0^2}\right\}\\
&= \frac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{1}{2}\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\left(\mu-\left(\frac{\sum_{i=1}^{N}x_i\sigma_0^2+\mu_0\sigma^2}{\sigma^2\sigma_0^2}\right)\left(\frac{N\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}\right)^{-1}\right)^2\right. && (9)\\
&+ \left.\frac{1}{2}\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\left(\left(\frac{\sum_{i=1}^{N}x_i\sigma_0^2+\mu_0\sigma^2}{\sigma^2\sigma_0^2}\right)\left(\frac{N\sigma_0^2+\sigma^2}{\sigma^2\sigma_0^2}\right)^{-1}\right)^2\right\} \exp\left\{\frac{\sum_{i=1}^{N}x_i^2}{\sigma^2}+\frac{\mu_0^2}{\sigma_0^2}\right\}\\
&= \frac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{1}{2}\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\left(\mu-\frac{\sum_{i=1}^{N}x_i\sigma_0^2+\mu_0\sigma^2}{N\sigma_0^2+\sigma^2}\right)^2\right\} && (10)\\
&\quad \exp\left\{\frac{1}{2}\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\left(\frac{\sum_{i=1}^{N}x_i\sigma_0^2+\mu_0\sigma^2}{N\sigma_0^2+\sigma^2}\right)^2+\frac{\sum_{i=1}^{N}x_i^2}{\sigma^2}+\frac{\mu_0^2}{\sigma_0^2}\right\}\\
&= C\exp\left\{-\frac{1}{2}\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\left(\mu-\frac{\sum_{i=1}^{N}x_i\sigma_0^2+\mu_0\sigma^2}{N\sigma_0^2+\sigma^2}\right)^2\right\} && (11)\\
&\propto \mathcal{N}\left(\mu\,\middle|\,\frac{\sum_{i=1}^{N}x_i\sigma_0^2+\mu_0\sigma^2}{N\sigma_0^2+\sigma^2},\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)^{-1}\right) && (12)
\end{aligned}
$$

- So, as shown above, the form of the posterior is also a Gaussian distribution.
- There are many conjugate prior relationships, e.g., Bernoulli-Beta, Gaussian-Gaussian, Gaussian-InverseWishart, Multinomial-Dirichlet

3 Bayesian Regression

- Look back our polynomial regression:

$$\min E^*(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n,\mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \tag{13}$$

This is equivalent to:

$$\max \prod_{n=1}^{N} \exp\left\{-\frac{1}{2}(y(x_n,\mathbf{w}) - t_n)^2\right\}\exp\left\{-\frac{\lambda}{2}\|\mathbf{w}\|_2^2\right\} \tag{14}$$

- As discussed, the first term is the likelihood and the second term is the prior on the weights
- These are Gaussian distributions:

$$\mathscr{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \tag{15}$$

- σ^2 is the variance OR $\frac{1}{\sigma^2}$ is the *precision*
- So, as λ gets big, variance gets smaller/tighter. As λ gets small, variance gets larger/wider.
- Previously, we used:

$$y = \sum_{j=0}^{M} w_j x^j \tag{16}$$

- We can extend this, to make it more general and flexible:

$$y = \sum_{j=0}^{M} w_j \phi_j(\mathbf{x}) \tag{17}$$

where $\phi_j(\mathbf{x})$ is a *basis function*
- For example:
  - Basis function we were using previously: $\phi_j(x) = x^j$ (for univariate x)
  - Linear Basis Function: $\phi_j(\mathbf{x}) = x_j$
  - Radial Basis Function: $\phi_j(\mathbf{x}) = \exp\left\{-\frac{(x-\mu_j)^2}{2s_j^2}\right\}$
  - Sigmoidal Basis Function: $\phi_j(\mathbf{x}) = \frac{1}{1+\exp\left\{\frac{\mathrm{x}-\mu_j}{s}\right\}}$
- As before:

$$t = y(\mathbf{x},\mathbf{w}) + \epsilon \tag{18}$$

- However, now:

$$y = \mathbf{w}^T\mathbf{\Phi}(\mathbf{x}) = [w_0, w_1, \ldots, w_M][\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \ldots, \phi_M(\mathbf{x})]^T \tag{19}$$

where $\epsilon \sim \mathscr{N}(\cdot|0,\beta^{-1})$

$$p(t|\mathbf{w},\beta) = \prod_{n=1}^{N}\mathscr{N}(t_n|\mathbf{w}^T\mathbf{\Phi}(\mathbf{x}_n),\beta^{-1}) \tag{20}$$

* So, what is the ‘’trick” to use to maximize this?

$$\mathscr{L} = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\beta E(\mathbf{w}) \tag{21}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{w}} = \beta\sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{\Phi}(\mathbf{x}_n))\mathbf{\Phi}(\mathbf{x}_n)^T = 0 \tag{22}$$

* This results in:

$$\mathbf{w}_{ML} = \left(\mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t} \tag{23}$$

where

$$\mathbf{\Phi} = [\mathbf{\Phi}(x_1), \mathbf{\Phi}(x_2), ...] \tag{24}$$

- What would you do if you want to include a prior? get the MAP solution? If assuming zero-mean Gaussian noise, then Regularized Least Squares!

3.0.1 Bayesian Linear Regression

- Recall: $E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$ where λ is the trade-off regularization parameter
- A simple regularizer (and the one we used previously) is: $E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$
- If we assume zero-mean Gaussian noise: $E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n)\right\}^2$
- Then, the total error becomes: $\frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n)\right\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$
- We can take the derivative, set it equal to zero and solve for the weights. When we do, we get:

$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{t} \tag{25}$$

- Recall, we can interpret this as:

$$\begin{aligned}
\min_{\mathbf{w}} E^* &= \min_{\mathbf{w}}\{E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})\} && (26)\\
&= \max_{\mathbf{w}}\{-E_D(\mathbf{w}) - \lambda E_W(\mathbf{w})\} && (27)\\
&= \max_{\mathbf{w}}\exp\{-E_D(\mathbf{w}) - \lambda E_W(\mathbf{w})\} && (28)\\
&= \max_{\mathbf{w}}\exp\{-E_D(\mathbf{w})\}\exp\{-\lambda E_W(\mathbf{w})\} && (29)\\
&\propto \max_{\mathbf{w}}\prod_{n=1}^{N}\mathscr{N}\left(t\left|\mathbf{w}^T\ (\mathbf{x}_n), \beta^{-1}\mathbf{I}\right.\right)\mathscr{N}\left(\mathbf{w}\left|\mathbf{m}_0, \mathbf{S}_0\right.\right) && (30)\\
&= \max_{\mathbf{w}} p\left(\mathbf{t}\left|\mathbf{w}, \mathbf{X}\right.\right)p\left(\mathbf{w}\right) && (31)\\
&\propto \max_{\mathbf{w}} p(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N) = \mathscr{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N) && (32)
\end{aligned}$$

where $\mathbf{m}_N = \mathbf{S}_N\left(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\mathbf{\Phi}^T\mathbf{t}\right)$ and $\mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} + \beta\mathbf{\Phi}^T\mathbf{\Phi}$

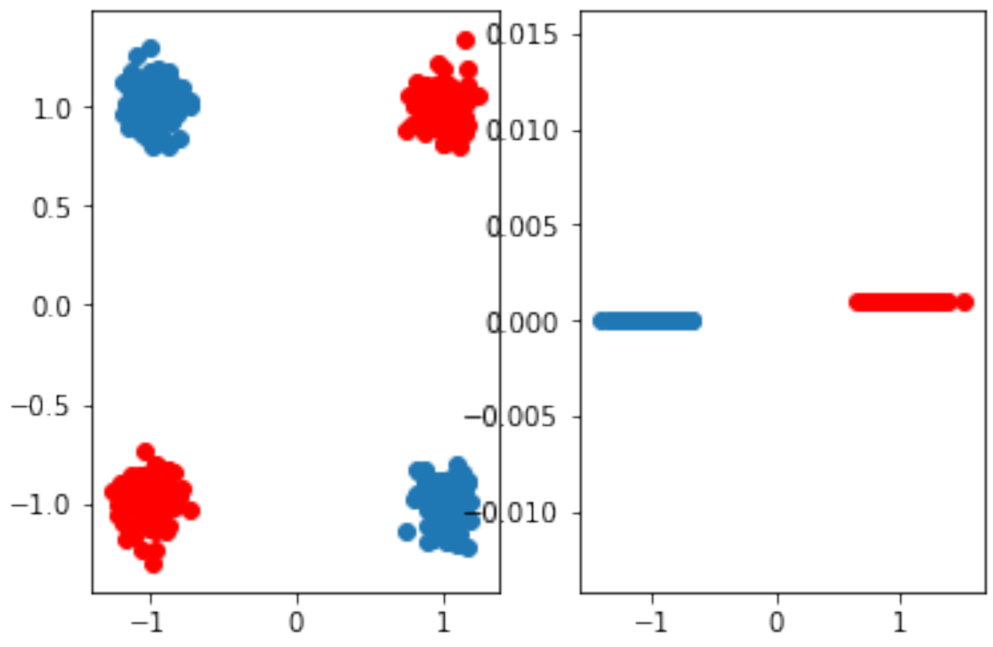
- What happens with different values of β and $\mathbf{S}_0$?
- To simplify, let us assume that $\mathbf{S}_0 = \alpha^{-1}\mathbf{I}$ and $\mathbf{m}_0 = \mathbf{0}$, thus, $\mathbf{m}_N = \beta\mathbf{S}_N\mathbf{\Phi}^T\mathbf{t}$ and $\mathbf{S}_N^{-1} = (\alpha^{-1}\mathbf{I})^{-1} + \beta\mathbf{\Phi}^T\mathbf{\Phi} = \alpha\mathbf{I} + \beta\mathbf{\Phi}^T\mathbf{\Phi}$

- This results in the following Log Posterior:

$$\ln p(\mathbf{w}|\mathbf{t}) = -\frac{\beta}{2}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\left(\mathbf{x}_n\right)\right)^2 - \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} + const \tag{33}$$

- Let us suppose we are dealing with 1-D data, $\mathbf{X} = \{x_1, \ldots, x_N\}$ and a linear form for y: $y(x, \mathbf{w}) = w_0 + w_1 x$
- We are going to generate synthetic data from: $t = -0.3 + 0.5x + \epsilon$ where ϵ is from zero-mean Gaussian noise. The goal is to estimate the true values $w_0 = -0.3$ and $w_1 = 0.5$.

[2]:
```
def innerBasisFunc(dataX):
        return np.vstack([dataX[i,:]@dataX[i,:].T for i in range(dataX.
 ↪shape[0])])

def prodBasisFunc(dataX):
        return np.vstack([dataX[i,0]*dataX[i,1] for i in range(dataX.shape[0])])

def xorExample():
        '''xorExample()'''
        class1X = np.vstack([np.array([-1,-1])+np.random.normal(0,.1,2 ) for i␣
 ↪in range(100)])
        class1X = np.vstack((class1X,np.vstack([np.array([1,1])+np.random.
 ↪normal(0,.1,2 ) for i in range(100)])))
        class2X = np.vstack([np.array([1,-1])+np.random.normal(0,.1,2 ) for i␣
 ↪in range(100)])
        class2X = np.vstack((class2X,np.vstack([np.array([-1,1])+np.random.
 ↪normal(0,.1,2 ) for i in range(100)])))
        phi1X = prodBasisFunc(class1X)
        phi2X = prodBasisFunc(class2X)

        fig = plt.figure()
        ax = fig.add_subplot(*[1,2,1])
        ax.scatter(class1X[:,0], class1X[:,1], c='r')
        ax.scatter(class2X[:,0], class2X[:,1])
        ax = fig.add_subplot(*[1,2,2])
        ax.scatter(phi1X, np.zeros(phi1X.shape)+0.001, c='r')
        ax.scatter(phi2X, np.zeros(phi2X.shape))
        plt.show()

xorExample()
```

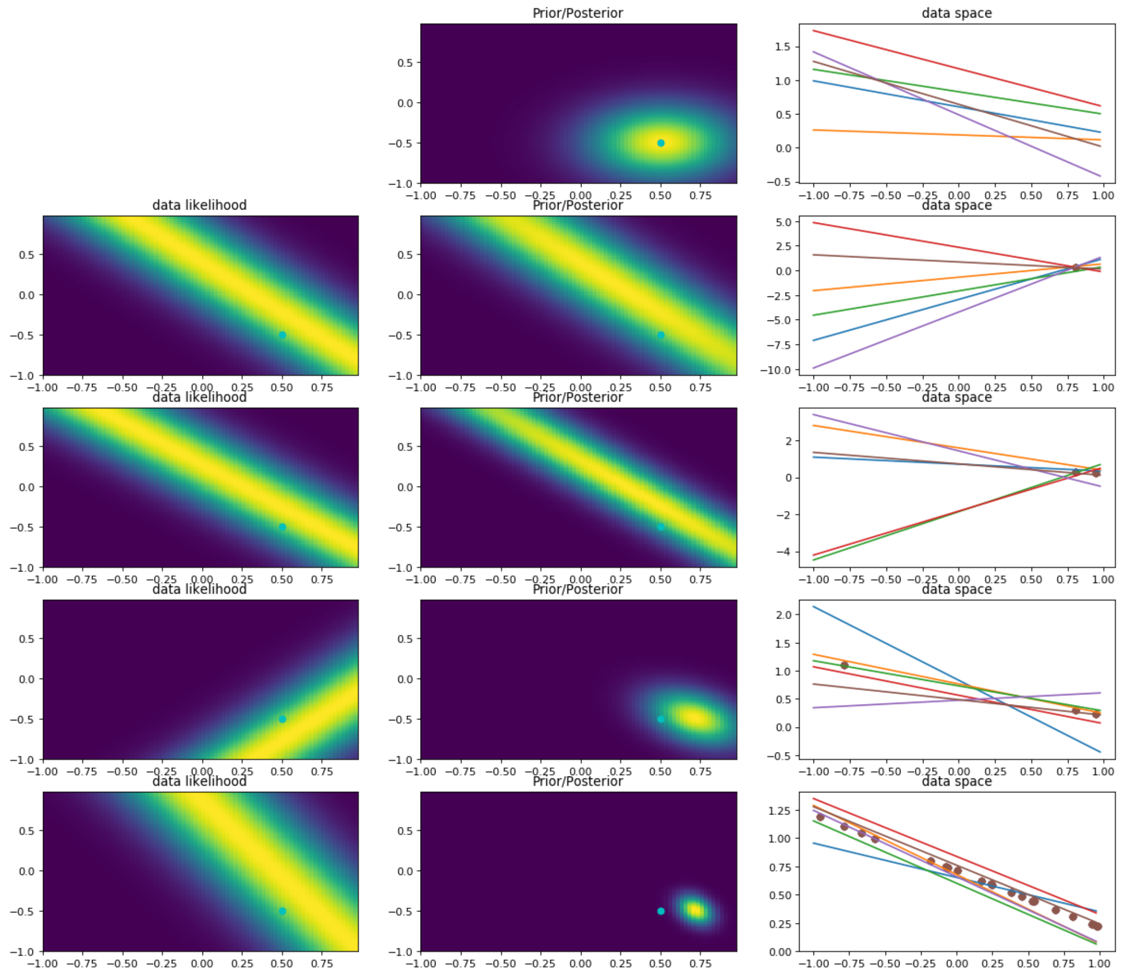


```
[12]: def likelihood_prior_func():
          fig = plt.figure(figsize=(18, 16), dpi= 80, facecolor='w', edgecolor='k')

          #set up variables
          a = 0.5
          b = -0.5
          rangeX = [-1, 1]
          step = 0.025
          X = np.mgrid[rangeX[0]:rangeX[1]:step]
          alpha = 10
          beta = 10
          S0 = (1/alpha)*np.eye(2)
          draw_num = (0,1,2,3,20)

          #initialize prior/posterior and sample data
          sigma = S0
          mean = [.5,-.5]
          draws = np.random.uniform(rangeX[0],rangeX[1],size=draw_num[-1])
          T = a + b*draws + np.random.normal(loc=0, scale=math.sqrt(1/beta))

          for i in range(len(draw_num)):
              if draw_num[i]>0: #skip first image
                  #Show data likelihood
                  Phi = np.vstack((np.ones(draws[0:draw_num[i]].shape), draws[0:
      ↪draw_num[i]]))
```

```
            t = T[0:draw_num[i]]
            sigma = np.linalg.inv(S0 + beta*Phi@Phi.T)
            mean = beta*sigma@Phi@t

            w0, w1 = np.mgrid[rangeX[0]:rangeX[1]:step, rangeX[0]:rangeX[1]:
↪step]
            p = multivariate_normal(t[draw_num[i]-1], 1/beta)
            out = np.empty(w0.shape)
            for j in range(len(w0)):
                out[j] = p.pdf(w0[j]+w1[j]*draws[draw_num[i]-1])

            ax = fig.add_subplot(*[len(draw_num),3,(i)*3+1])
            ax.pcolor(w0, w1, out)
            ax.scatter(a,b, c='c')
            myTitle = 'data likelihood'
            ax.set_title("\n".join(textwrap.wrap(myTitle, 100)))

        #Show prior/posterior
        w0, w1 = np.mgrid[rangeX[0]:rangeX[1]:step, rangeX[0]:rangeX[1]:step]
        pos = np.empty(w1.shape + (2,))
        pos[:, :, 0] = w0; pos[:, :, 1] = w1
        p = multivariate_normal(mean, sigma)

        ax = fig.add_subplot(*[len(draw_num),3,(i)*3+2])
        ax.pcolor(w0, w1, p.pdf(pos))
        ax.scatter(a,b, c='c')
        myTitle = 'Prior/Posterior'
        ax.set_title("\n".join(textwrap.wrap(myTitle, 100)))

        #Show data space
        for j in range(6):
            w0, w1 = np.random.multivariate_normal(mean, sigma)
            t = w0 + w1*X
            ax = fig.add_subplot(*[len(draw_num),3,(i)*3+3])
            ax.plot(X,t)
            if draw_num[i] > 0:
                ax.scatter(Phi[1,:], T[0:draw_num[i]])
            myTitle = 'data space'
            ax.set_title("\n".join(textwrap.wrap(myTitle, 100)))

    plt.show()

likelihood_prior_func()
```

```
/Users/alinazare/anaconda3/lib/python3.7/site-
packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning:
Adding an axes using the same arguments as a previous axes currently reuses the
```

```
earlier instance.  In a future version, a new instance will always be created
and returned.  Meanwhile, this warning can be suppressed, and the future
behavior ensured, by passing a unique label to each axes instance.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



```
[ ]:
```