Advanced Introduction to Machine Learning
— Spring Quarter, Week 1 —
https://canvas.uw.edu/courses/1372141

Prof. Jeff Bilmes

University of Washington, Seattle
Departments of: Electrical & Computer Engineering, Computer Science & Engineering
http://melodi.ee.washington.edu/~bilmes

March 30th, 2020
Welcome!

- Welcome to the class: Advanced Introduction to Machine Learning
- An advanced introduction to machine learning and its applications in data science and artificial intelligence.
Advanced Introduction to Machine Learning

- Introduction: assumes little background ML knowledge
- Advanced: requires good CS & Math and programming (mostly python, maybe C++) background. Also, there is a lot of material to cover!! Ambitious “(of a plan or piece of work) intended to satisfy high aspirations and therefore difficult to achieve.”
- Machine Learning: what this class is about.
Logistics

- Mon/Wed 10:30-12:30 via zoom.
- Weekly virtual evening office hours: Thursdays, 10:00-11:00pm, via zoom (same link).
- Our web page (https://canvas.uw.edu/courses/1372141), lecture slides to appear both before and after lecture.
- Use our discussion board (https://canvas.uw.edu/courses/1372141/discussion_topics) for all questions, comments, so that all will benefit from them being answered. Please use that rather than email.
Teaching Assistant

- Ricky Zhang <yz325@uw.edu>
Facts about the class

- **Prerequisites:** knowledge in probability, statistics, linear algebra, multivariate calculus, some information theory, mathematical optimization

- **Text:** We will be using class slides and drawing from material mostly available for free on the web although there are some recommended books as well (see other slide on references).

- **Grades and Assignments:** We will have approximately one assignment every two weeks. Each assignment is combination of math problems and large programming project(s) (in python). Grades are based on these assignments.

- **No in-class tests, no final exam.** Grade is based entirely on quality, clarity, and correctness in your HW problem sets and python ML projects.
More Facts about the class

- **COLLABORATION POLICY:** Homework must be done individually: each student must hand in their own answers. In addition, each student must write and submit their own code in the programming part of the assignment (we will run and test your code). It is acceptable, however, for students to collaborate in figuring out answers and helping each other solve the problems (but please use our assignment dropbox (https://canvas.uw.edu/courses/1372141/assignments)). You must also indicate on each homework with whom you collaborated.

- Homework assigned asynchronously via canvas, must be submitted electronically using our assignment dropbox (https://canvas.uw.edu/courses/1372141/assignments). Submissions: Single PDF file and, when relevant, .zip files with code/data/ipynb. Photos of very neatly hand written solutions, combined into one PDF, are fine.

- Lecture slides will appear on canvas before the class begins, and updates post-lecture will also be posted with markups, as well as a youtube recording of each lecture.
Homework late policy

- For every additional day you take for the submission, we would subtract 5% of your grade.
- This continues for a week, after which we will stop accepting submissions and release the solutions.
- Example: EffectiveGrade = OriginalGrade × (1 − LateDays × 0.05)

\[\text{where } 0 \leq \text{LateDays} \leq 7.\]

- If the homework is due on the 6th and you submit on the morning of 7th, you’ll get max 95% of your grade, and so on up till 11:59 on the 13th – your last day to submit – when you will get max 65% of your grade.
- We recommend that you submit however much you have completed by the deadline.
- If you resubmit your assignment after the deadline, please make note of which problem(s) you have updated; we will only apply the reduction to that problem (entirely, not the subproblems).
- Also, note that we will not be giving any Late Days for the last homework, since there would be a very short window for us to grade it.
Washington state law requires that UW develop a policy for accommodation of student absences or significant hardship due to reasons of faith or conscience, or for organized religious activities. The UW’s policy, including more information about how to request an accommodation, is available at Religious Accommodations Policy (https://registrar.washington.edu/staffandfaculty/religious-accommodations-policy/). Accommodations must be requested within the first two weeks of this course using the Religious Accommodations Request form (https://registrar.washington.edu/students/religious-accommodations-request/).
Other logistics

- Almost all equations will have numbers.
- Equations will be numbered with lecture number, and number within lecture in the form \((\ell.j)\) where \(\ell\) is the lecture number and \(j\) is the \(j^{th}\) equation in lecture \(\ell\). For example,

\[
L(w) = \sum_{i=1}^{n} (y_i - x_i^T w)^2 + \lambda \| w \|^2_2 \tag{1.1}
\]

- Theorems, Lemmas, postulates, etc. will be numbered with \((\ell.s.j)\) where \(\ell\) is the lecture number, \(s\) is the section number, and \(j\) is the order within that section.

**Theorem 1.1.1 (foo’s theorem)**

**foo**

- Exception to these rules is in the review sections, where theorems, equation, etc. (even if repeated) will have new reference numbers.
Class Road Map

- W1(3/30,4/1): What is ML, Probability, Coins, Gaussians and linear regression, Associative Memories, Supervised Learning
- W2(4/6,4/8): More supervised, logistic regression, complexity and bias/variance tradeoff
- W6(5/4,5/6): Curse of Dimensionality, Dimensionality Reduction, $k$-NN
- W7(5/11,5/13): $k$-NN, LSH, DTs, Bootstrap/Bagging, Boosting & Random Forests, GBDTs
- W8(5/18,5/20): Graphs; Graphical Models (Factorization, Inference, MRFs, BNs);
- W9(5/27,6/1): Learning Paradigms; Clustering; EM Algorithm;
- W10(6/3,6/8): Spectral Clustering, Graph SSL, Deep models, (SVMs, RL); The Future.

Last lecture is 6/8 since 5/25 is holiday (or we could just have lecture on 5/25).
Great Interest in Machine Learning/AI

see more at https://nips.cc/Conferences/2018/Sponsors
Class Description

This is an ambitious class that will provide a broad overview of a large variety of machine learning methods in a short amount of time. You will learn to understand the basics of: linear regression; logistic regression; k-nearest neighbors; PCA, LDA, and dimensionality reduction methods; feature selection and engineering; cross validation; the bootstrap, bagging, and boosting; decision trees and random forests; naive Bayes; generative vs. discriminative models; support vector machines and kernel methods; neural networks; Bayesian nonparametric methods; clustering; ensemble methods; reinforcement learning; representation learning; information theory; Gaussian processes; supervised, unsupervised, and semi-supervised learning; graphical models; sparsity and compressed sensing; planning and control; information retrieval; structured prediction; matrix factorization; Monte Carlo methods; time-series analysis and HMMs; multi-agent learning; transfer and multi-task learning; active learning; submodularity; and machine teaching. Along the way, we will motivate the above using applications in computational biology, networks, computer vision, speech recognition, and natural language processing. We will also touch on the philosophy of machine learning and artificial intelligence, and discuss if we can build a computer program having artificial general intelligence. The class will require programming in python and the use of python libraries (e.g., numpy, sklearn, and pytorch). Previous knowledge of linear algebra, calculus, and basic probability theory and statistics is a must.
1. Introduction
• What is ML
• What is AI
• Why are we so interested in these topics right now?

2. ML Paradigms/Concepts
• Overfitting/Underfitting, model complexity, bias/variance
• size of data, big data, sample complexity
• ERM, loss + regularization, loss functions, regularizers
• Supervised, unsupervised, and semi-supervised learning;
  • reinforcement learning, RL, multi-agent, planning/control
• transfer and multi-task learning
• active learning
• machine teaching

3. Dealing with Features
• PCA, LDA, MDS, T-SNE, and dimensionality reduction methods
• Locality sensitive hashing (LSH)
• feature selection
• feature engineering
• matrix factorization & feature engineering
• representation learning

4. Evaluation
• accuracy/error, precision/recall, ROC, likelihood/posterior, cost/utility, margin
• train/eval/test data splits
• n-fold cross validation
• method of the bootstrap

5. Optimization Methods
• Unconstrained Continuous Optimization: (stochastic) gradient descent (SGD), adaptive learning rates, conjugate gradient, 2nd order Newton
• Constrained Continuous Optimization: Frank-Wolf (conditional gradient descent), projected gradient, linear, quadratic, and convex programming
• Discrete optimization - greedy, beam search, branch-and-bound, submodular optimization.

6. Inference Methods
• probabilistic inference
• MLE, MAP
• belief propagation
• forward/backpropagation
• Monte Carlo methods

7. Models & Representation
• linear least squares, linear regression, logistic regression, sparsity, ridge, lasso
• generative vs. discriminative models
• Naive Bayes
• k-nearest neighbors
• clustering, k-means, k-mediods, EM & GMMs, single linkage
• decision trees and random forests
• support vector machines, kernel methods, max margin
• perceptron, neural networks, DNNs
• Gaussian processes
• Bayesian nonparametric methods
• ensemble methods
• the bootstrap, bagging, and boosting
• graphical models
• time-series, HMMs, DBNs, RNNs
• structured prediction
• grammars (as in NLP)

8. Philosophy, Humanity, Spirituality
• artificial intelligence (AI)
• artificial general intelligence (AGI)
• artificial intelligence vs. science fiction

9. Applications
• computational biology
• social networks
• computer vision
• speech recognition
• natural language processing
• information retrieval
• collaborative filtering/matrix factorization

10. Programming
• python
• libraries (e.g., NumPy, SciPy, matplotlib, scikit-learn (sklearn), pytorch, CNTK, Theano, tensorflow, keras, H2O, etc.
• HPC: C/C++, CUDA, vector processing

11. Background
• linear algebra
• multivariate calculus
• probability theory and statistics
• information theory
• mathematical (e.g., convex) optimization

12. Other Techniques
• compressed sensing
• submodularity, diversity/homogeneity modeling

1.2 Determinant
Let A be an $n \times n$ matrix.
$$\det(A) = \prod \lambda_i, \quad \lambda_i = \sigma(A)$$
$$\det(A^T) = \det(A), \quad \text{if } A \in \mathbb{R}^{n\times n}$$
$$\det(A^T) = \det(A)$$
$$\det(A B) = \det(A) \det(B)$$
$$\det(A^{-1}) = 1/\det(A)$$
$$\det(A^T) = \det(A)^T$$
$$\det(I + uv^T) = 1 + u^T v$$
Machine Learning (Grain of Salt) Cheat Sheet

Unsupervised Learning: Clustering

- k-means
- k-modes
- Gaussian Mixture Model
- Prefer Probability
- Categorical Variables
- Need to Specify k
- Hierarchical
- DBSCAN

START

Unsupervised Learning: Dimension Reduction

- Dimension Reduction
- Topic Modeling
- Probabilistic
- Principal Component Analysis
- Singular Value Decomposition

Supervised Learning: Classification

- Linear SVM
- Naive Bayes
- Data Is Too Large
- Explainable
- Speed or Accuracy
- Predicting Numeric
- Naive Bayes
- Decision Tree
- Kernel SVM
- Random Forest
- Neural Network
- Gradient Boosting Tree

Supervised Learning: Regression

- Speed or Accuracy
- Decision Tree
- Linear Regression
- Random Forest
- Neural Network
- Gradient Boosting Tree

from https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/

Prof. Jeff Bilmes
EE511/Spring 2020/Adv. Intro ML - Week 1 - March 30th, 2020
Recommended References

- These class slides.
- Free resources

https://towardsdatascience.com/list-of-free-must-read-machine-learning-books-89576749d2ff
What is Machine Learning and Machine Intelligence?

Humans: not smart enough to directly program complex tasks.
Production of algorithms that, rather than directly human written, are indirectly produced via mathematical optimization parameterized by data.

"All problems in computer science can be solved by another level of indirection" David Wheeler.

IA: Indirect Algorithms

Grand challenges: education, poverty, energy/climate change, and health.
What is Machine Learning and Machine Intelligence?

Humans: not smart enough to directly program complex tasks.
Machine Learning and Machine Intelligence

- What is Machine Learning and Machine Intelligence?
- Humans: not smart enough to directly program complex tasks.
- Production of algorithms that, rather than directly human written, are indirectly produced via mathematical optimization parameterized by data.
What is Machine Learning and Machine Intelligence?

Humans: not smart enough to directly program complex tasks.

Production of algorithms that, rather than directly human written, are indirectly produced via mathematical optimization parameterized by data.

“All problems in computer science can be solved by another level of indirection” David Wheeler.
Machine Learning and Machine Intelligence

- What is Machine Learning and Machine Intelligence?
- Humans: not smart enough to directly program complex tasks.
- Production of algorithms that, rather than directly human written, are indirectly produced via mathematical optimization parameterized by data.
- “All problems in computer science can be solved by another level of indirection” David Wheeler.

IA: Indirect Algorithms
Machine Learning and Machine Intelligence

- What is Machine Learning and Machine Intelligence?
- Humans: not smart enough to directly program complex tasks.
- Production of algorithms that, rather than directly human written, are indirectly produced via mathematical optimization parameterized by data.
- “All problems in computer science can be solved by another level of indirection” David Wheeler.
- IA: Indirect Algorithms
- Grand challenges: education, poverty, energy/climate change, and health.
Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do. – Donald Knuth
Machine Learning is the art of repeatedly telling a computer what one wants the computer to tell a second computer about a lot of data. This continues until the second computer gets it right.
Traditional Computer Programming vs. ML

https://marticus.org/what-is-machine-learning-and-does-it-matter/

The Ideal Machine Learning Methods

- Simple to define
- Naturally suited to many real-world applications
- Mathematically expressive
- Efficient & scalable to large problem instances
Object Recognition/Classification

- The goal is to make a distinction between objects in real world.
Object Recognition/Classification

- The goal is to make a distinction between objects in real world.
- Humans easily do it every day and find it easy.
Object Recognition/Classification

- The goal is to make a distinction between objects in real world.
- Humans easily do it every day and find it easy.
- For example, what are the following objects:
Object Recognition/Classification

- The goal is to make a distinction between objects in real world.
- Humans easily do it every day and find it easy.
- For example, what are the following objects:

![Chairs](image1)

- And what are the following objects:

![Tables](image2)
Object Recognition/Classification

- How to get a computer to do it? What is difference between a table and a chair?
Object Recognition/Classification

- How to get a computer to do it? What is difference between a table and a chair?
- When is it not a chair and a table?
Object Recognition/Classification

- How to get a computer to do it? What is difference between a table and a chair?
- When is it not a chair and a table? What about:

![Image of a chair, a small table, and a massage chair]

*Euro 35*
Object Recognition/Classification

- How to get a computer to do it? What is difference between a table and a chair?
- When is it not a chair and a table? What about:
  - Sometimes it is easy.
Object Recognition/Classification

- How to get a computer to do it? What is difference between a table and a chair?

- When is it not a chair and a table? What about:

  ![Chair Images]

- Sometimes it is easy. Sometimes it is not so easy:
Object Recognition

- Sometimes it is a continuum (Escher, Liberation, 1955)
Object Recognition

- Sometimes it is a continuum (Escher, Liberation, 1955)
- What is foreground vs. background? (Escher, Mosaic, 1957)
Probability and Uncertainty

Key point: the world is a complicated place, we cannot know everything, and even what we think we know we can (nor should) not always be certain. Uncertainty abounds!
Probability and Uncertainty

- Key point: the world is a complicated place, we cannot know everything, and even what we think we know we can (nor should) not always be certain. **Uncertainty abounds!**
- Need a representation of uncertainty.
Probability and Uncertainty

- Key point: the world is a complicated place, we cannot know everything, and even what we think we know we can (nor should) not always be certain. **Uncertainty abounds!**
- Need a representation of uncertainty.
- Probability has a precise mathematical definition (Kolmogorov axioms), but we use it in deference to the inevitable uncertainty surrounding all decisions.
Probability and Uncertainty

- Key point: the world is a complicated place, we cannot know everything, and even what we think we know we can (nor should) not always be certain. **Uncertainty abounds!**
- Need a representation of uncertainty.
- Probability has a precise mathematical definition (Kolmogorov axioms), but we use it in deference to the inevitable uncertainty surrounding all decisions.
- **Simple and subjective working definition:**

\[
\text{probability} = \frac{\text{number of cases something happened}}{\text{number of total cases}}.
\]  

(1.2)
Probability and Uncertainty

Key point: the world is a complicated place, we cannot know everything, and even what we think we know we can (nor should) not always be certain. **Uncertainty abounds!**

Need a representation of uncertainty.

Probability has a precise mathematical definition (Kolmogorov axioms), but we use it in deference to the inevitable uncertainty surrounding all decisions.

Simple and subjective working definition:

\[
\text{probability} = \frac{\text{number of cases something happened}}{\text{number of total cases}}.
\]  

Good for repeatable measurable events (e.g., coins flips, dice, etc.).
Probability and Uncertainty

- Key point: the world is a complicated place, we cannot know everything, and even what we think we know we can (nor should) not always be certain. **Uncertainty abounds!**
- Need a representation of uncertainty.
- Probability has a precise mathematical definition (Kolmogorov axioms), but we use it in deference to the inevitable uncertainty surrounding all decisions.
- Simple and subjective working definition:

\[
\text{probability} = \frac{\text{number of cases something happened}}{\text{number of total cases}}.
\]  
(1.2)

- Good for repeatable measurable events (e.g., coins flips, dice, etc.).
- Harder for future events (probability it will rain tomorrow, probability Manchester City wins Liverpool, etc.).
Probability and Uncertainty

- Key point: the world is a complicated place, we cannot know everything, and even what we think we know we can (nor should) not always be certain. **Uncertainty abounds!**
- Need a representation of uncertainty.
- Probability has a precise mathematical definition (Kolmogorov axioms), but we use it in deference to the inevitable uncertainty surrounding all decisions.
- Simple and subjective working definition:

\[
\text{probability} = \frac{\text{number of cases something happened}}{\text{number of total cases}}. \tag{1.2}
\]

- Good for repeatable measurable events (e.g., coins flips, dice, etc.).
- Harder for future events (probability it will rain tomorrow, probability Manchester City wins Liverpool, etc.).
- Despite shortcomings, used as representation of uncertainty/certainty (i.e., probability that image \(x\) contains face of person \(y\)).
Probability and Uncertainty

- Key point: the world is a complicated place, we cannot know everything, and even what we think we know we can (nor should) not always be certain. Uncertainty abounds!
- Need a representation of uncertainty.
- Probability has a precise mathematical definition (Kolmogorov axioms), but we use it in deference to the inevitable uncertainty surrounding all decisions.
- Simple and subjective working definition:

\[
\text{probability} = \frac{\text{number of cases something happened}}{\text{number of total cases}}. \tag{1.2}
\]

- Good for repeatable measurable events (e.g., coins flips, dice, etc.).
- Harder for future events (probability it will rain tomorrow, probability Manchester City wins Liverpool, etc.).
- Despite shortcomings, used as representation of uncertainty/certainty (i.e., probability that image \( x \) contains face of person \( y \)).
- Machine learning often strives for the “best” probabilities in data using learning algorithms.
Coin Flipping and ML

- \( D = \{b_1, b_2, \ldots, b_n\} \) is series of \( n \) independent and identical coin flips, \( b_i \in \{H, T\} \).
Coin Flipping and ML

- $D = \{b_1, b_2, \ldots, b_n\}$ is a series of $n$ independent and identical coin flips, $b_i \in \{H, T\}$.
- $k = |\{i : b_i = H\}|$ is the count of the number of heads in $D$.
Coin Flipping and ML

- \( \mathcal{D} = \{b_1, b_2, \ldots, b_n\} \) is series of \( n \) independent and identical coin flips, \( b_i \in \{H, T\} \).
- \( k = |\{i : b_i = H\}| \) is the count of the number of heads in \( \mathcal{D} \).
- How true, or likely, is it that \( \theta \) is probability of heads?

\[
\Pr(\mathcal{D}|\theta) = \theta^k (1 - \theta)^{n-k} = \text{Likelihood of } \mathcal{D} \text{ given } \theta \tag{1.3}
\]
Coin Flipping and ML

- $\mathcal{D} = \{b_1, b_2, \ldots, b_n\}$ is series of $n$ independent and identical coin flips, $b_i \in \{H, T\}$.
- $k = |\{i : b_i = H\}|$ is the count of the number of heads in $\mathcal{D}$
- How true, or likely, is it that $\theta$ is probability of heads?

$$\Pr(\mathcal{D} | \theta) = \theta^k (1 - \theta)^{n-k} = \text{Likelihood of } \mathcal{D} \text{ given } \theta \quad (1.3)$$

- How to find the most likely explanation of $\mathcal{D}$? Maximum likelihood

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta \in [0,1]} \Pr(\mathcal{D} | \theta) = \arg\max_{\theta \in [0,1]} \log \Pr(\mathcal{D} | \theta) \quad (1.4)$$
Coin Flipping and ML

- $\mathcal{D} = \{b_1, b_2, \ldots, b_n\}$ is series of $n$ independent and identical coin flips, $b_i \in \{\text{H}, \text{T}\}$.

- $k = |\{i : b_i = \text{H}\}|$ is the count of the number of heads in $\mathcal{D}$

- How true, or likely, is it that $\theta$ is probability of heads?

$$
\Pr(\mathcal{D}|\theta) = \theta^k (1 - \theta)^{n-k} = \text{Likelihood of } \mathcal{D} \text{ given } \theta \quad (1.3)
$$

- How to find the most likely explanation of $\mathcal{D}$? Maximum likelihood

$$
\hat{\theta}_{\text{MLE}} = \arg\max_{\theta \in [0, 1]} \Pr(\mathcal{D}|\theta) = \arg\max_{\theta \in [0, 1]} \log \Pr(\mathcal{D}|\theta) \quad (1.4)
$$

- How to find $\hat{\theta}_{\text{MLE}}$, calculus, $\frac{\partial}{\partial \theta} \log \Pr(\mathcal{D}|\theta) = 0$ leads to

$$
\hat{\theta}_{\text{MLE}} = \frac{k}{n} \quad (1.5)
$$
Coin Flipping and ML

- $\mathcal{D} = \{b_1, b_2, \ldots, b_n\}$ is series of $n$ independent and identical coin flips, $b_i \in \{\text{H}, \text{T}\}$.
- $k = |\{i : b_i = \text{H}\}|$ is the count of the number of heads in $\mathcal{D}$
- How true, or likely, is it that $\theta$ is probability of heads?

$$
\Pr(\mathcal{D}|\theta) = \theta^k (1 - \theta)^{n-k} = \text{Likelihood of } \mathcal{D} \text{ given } \theta
$$ (1.3)

- How to find the most likely explanation of $\mathcal{D}$? Maximum likelihood

$$
\hat{\theta}_{\text{MLE}} = \arg\max_{\theta \in [0,1]} \Pr(\mathcal{D}|\theta) = \arg\max_{\theta \in [0,1]} \log \Pr(\mathcal{D}|\theta)
$$ (1.4)

- How to find $\hat{\theta}_{\text{MLE}}$, calculus, $\frac{\partial}{\partial \theta} \log \Pr(\mathcal{D}|\theta) = 0$ leads to

$$
\hat{\theta}_{\text{MLE}} = \frac{k}{n}
$$ (1.5)

- Thus, computing $k$ and dividing by $n$ is a simple way to learn!
Random Learnt Model

- Any function of a random variable is also potentially random.
Random Learnt Model

- Any function of a random variable is also potentially random.
- $D$ is a random sample, so our estimate $\hat{\theta}_{\text{MLE}}(D)$ is also random, as it is a function of random sample $D$. 

Since $\hat{\theta}_{\text{MLE}}(D)$ is random, we can measure the probability that it deviates from truth. Let $\theta^*$ be the true parameter, than Hoeffding’s inequality states that:

$$\Pr(|\hat{\theta}_{\text{MLE}}(D) - \theta^*| \geq \epsilon) \leq 2e^{-\frac{2n\epsilon^2}{2}} \quad (1.6)$$

This gets really good quickly as $n$ (number of flips) gets large. Large data sets lead to better (or at least no worse) learning!

“There’s no data like more data”, and big data is pretty good, especially with lots of GPUs!

In general, concentration inequalities (such as Markov’s, Chebyshev’s, Chernoff’s, Hoeffding’s, Bennet/Bernstein’s, etc.) are useful to understanding properties of how quickly learning takes place. E.g., PAC learning is a form that allow us to compute the probability that a learnt model deviates from the true model.
Any function of a random variable is also potentially random.

\( \mathcal{D} \) is a random sample, so our estimate \( \hat{\theta}_{\text{MLE}}(\mathcal{D}) \) is also random, as it is a function of random sample \( \mathcal{D} \).

Since \( \hat{\theta}_{\text{MLE}}(\mathcal{D}) \) is random, we can measure the probability that it deviates from truth. Let \( \theta^* \) be the true parameter, than Hoeffding’s inequality states that:

\[
\Pr(|\hat{\theta}_{\text{MLE}}(\mathcal{D}) - \theta^*| \geq \epsilon) \leq 2e^{-2n\epsilon^2}
\]  

(1.6)
Any function of a random variable is also potentially random.

$\mathcal{D}$ is a random sample, so our estimate $\hat{\theta}_{\text{MLE}}(\mathcal{D})$ is also random, as it is a function of random sample $\mathcal{D}$.

Since $\hat{\theta}_{\text{MLE}}(\mathcal{D})$ is random, we can measure the probability that it deviates from truth. Let $\theta^*$ be the true parameter, than Hoeffding’s inequality states that:

$$\Pr(|\hat{\theta}_{\text{MLE}}(\mathcal{D}) - \theta^*| \geq \epsilon) \leq 2e^{-2n\epsilon^2}$$  \hspace{1cm} (1.6)

This gets really good quickly as $n$ (number of flips) gets large. Large data sets lead to better (or at least no worse) learning!
Random Learnt Model

- Any function of a random variable is also potentially random.
- $\mathcal{D}$ is a random sample, so our estimate $\hat{\theta}_{\text{MLE}}(\mathcal{D})$ is also random, as it is a function of random sample $\mathcal{D}$.
- Since $\hat{\theta}_{\text{MLE}}(\mathcal{D})$ is random, we can measure the probability that it deviates from truth. Let $\theta^*$ be the true parameter, than Hoeffding’s inequality states that:

$$\Pr(|\hat{\theta}_{\text{MLE}}(\mathcal{D}) - \theta^*| \geq \epsilon) \leq 2e^{-2n\epsilon^2} \quad (1.6)$$

- This gets really good quickly as $n$ (number of flips) gets large. Large data sets lead to better (or at least no worse) learning!
- “There’s no data like more data”, and big data is pretty good, especially with lots of GPUs!
Any function of a random variable is also potentially random.

$\mathcal{D}$ is a random sample, so our estimate $\hat{\theta}_{\text{MLE}}(\mathcal{D})$ is also random, as it is a function of random sample $\mathcal{D}$.

Since $\hat{\theta}_{\text{MLE}}(\mathcal{D})$ is random, we can measure the probability that it deviates from truth. Let $\theta^*$ be the true parameter, than Hoeffding’s inequality states that:

$$\Pr(|\hat{\theta}_{\text{MLE}}(\mathcal{D}) - \theta^*| \geq \epsilon) \leq 2e^{-2n\epsilon^2} \quad (1.6)$$

This gets really good quickly as $n$ (number of flips) gets large. Large data sets lead to better (or at least no worse) learning!

“There’s no data like more data”, and big data is pretty good, especially with lots of GPUs!

In general, concentration inequalities (such as Markov’s, Chebyshev’s, Chernoff’s, Hoeffding’s, Bennet/Bernstein’s, etc.) are useful to understanding properties of how quickly learning takes place. E.g., PAC learning is a form that allow us to compute the probability that a learnt model deviates from the true model.
Multivariate Gaussian (Normal), and samples thereof

- \( \mathcal{D} = \{x_1, x_2, \ldots, x_n\} \) is series of \( n \) independent and identically distributed (i.i.d.) \( m \)-dimensional real-valued samples \( \forall i, x_i \in \mathbb{R}^m \) from a Gaussian (or normal) distribution.

\[
x_i \sim \text{Pr}(x|\mu, C) = \frac{1}{|2\pi C|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu)^\top C^{-1} (x - \mu) \right)
\]

where \( \mu \in \mathbb{R}^m \) is a mean vector and \( C \) is \( m \times m \) a positive definite covariance matrix.
Multivariate Gaussian (Normal), and samples thereof

- \( D = \{x_1, x_2, \ldots, x_n\} \) is series of \( n \) independent and identically distributed (i.i.d.) \( m \)-dimensional real-valued samples \( \forall i, x_i \in \mathbb{R}^m \) from a Gaussian (or normal) distribution

\[
x_i \sim \Pr(x|\mu, C) = \frac{1}{|2\pi C|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu)^\top C^{-1} (x - \mu) \right)
\]

(1.7)

where \( \mu \in \mathbb{R}^m \) is a mean vector and \( C \) is \( m \times m \) a positive definite covariance matrix. Notationally, we often say \( x \sim \mathcal{N}(x; \mu, C) \).
Multivariate Gaussian (Normal), and samples thereof

- $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$ is series of $n$ independent and identically distributed (i.i.d.) $m$-dimensional real-valued samples $\forall i, x_i \in \mathbb{R}^m$ from a Gaussian (or normal) distribution

\[
x_i \sim \Pr(x|\mu, C) = \frac{1}{|2\pi C|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu)^\top C^{-1} (x - \mu) \right)
\]  

(1.7)

where $\mu \in \mathbb{R}^m$ is a mean vector and $C$ is $m \times m$ a positive definite covariance matrix. Notationally, we often say $x \sim \mathcal{N}(x; \mu, C)$.

- Symmetric matrix $C \in \mathbb{R}^{m \times m}$ is positive definite ($\in \mathbb{P}(m)$) if all of its Eigenvalues are positive (alternatively $\langle x, Cx \rangle > 0, \forall x \neq 0$).
Multivariate Gaussian (Normal), and samples thereof

- $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$ is series of $n$ independent and identically distributed (i.i.d.) $m$-dimensional real-valued samples $\forall i, x_i \in \mathbb{R}^m$ from a Gaussian (or normal) distribution

$$x_i \sim \Pr(x|\mu, C) = \frac{1}{|2\pi C|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu)^\top C^{-1} (x - \mu) \right)$$

where $\mu \in \mathbb{R}^m$ is a mean vector and $C$ is $m \times m$ a positive definite covariance matrix. Notationally, we often say $x \sim \mathcal{N}(x; \mu, C)$.

- Symmetric matrix $C \in \mathbb{R}^{m \times m}$ is positive definite ($\in \mathcal{P}(m)$) if all of its Eigenvalues are positive (alternatively $\langle x, Cx \rangle > 0, \forall x \neq 0$).

- Example sample of a 2D Gaussian.
Simple python code for sampling Gaussians

```
import numpy as np
import matplotlib.pyplot as plt

mean = [0, 0]
cov = [[1, 1.5], [1.5, 10]]
X = np.random.multivariate_normal(mean, cov, 1000)

fig, ax = plt.subplots(figsize=(10, 10))
plt.scatter(X[:,0], X[:,1], c='r')
plt.grid()
plt.show()
fig.savefig("Gaussian_2D.pdf", bbox_inches='tight')
```

Useful environment for testing python: https://jupyter.org
Learning Gaussians

Given the data sample $\mathcal{D}$ without knowing $\mu, C$, how likely is the sample under some hypothesized parameters $\tilde{\mu}, \tilde{C}$.

\[
\log \Pr(\mathcal{D}|\tilde{\mu}, \tilde{C}) = \sum_{i=1}^{n} \log \Pr(x_i|\tilde{\mu}, \tilde{C}) \equiv \log \text{Likelihood of } \mathcal{D} \text{ given } \tilde{\mu}, \tilde{C}
\] (1.8)

To find MLE quantities, again calculus, $\frac{\partial}{\partial \mu} \log \Pr(\mathcal{D}|\mu, C) = 0$ and $\frac{\partial}{\partial C} \log \Pr(\mathcal{D}|\mu, C) = 0$ leads to

$\mu_{\text{MLE}} = \frac{1}{n} \sum_{i=1}^{n} x_i$ and $C_{\text{MLE}} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_{\text{MLE}})(x_i - \mu_{\text{MLE}})^\top$ (1.11)
Learning Gaussians

• Given the data sample $\mathcal{D}$ without knowing $\mu, C$, how likely is the sample under some hypothesized parameters $\tilde{\mu}, \tilde{C}$.

$$
\log \Pr(\mathcal{D}|\tilde{\mu}, \tilde{C}) = \sum_{i=1}^{n} \log \Pr(x_i|\tilde{\mu}, \tilde{C})
$$

$$
\triangleq \log \text{Likelihood of } \mathcal{D} \text{ given } \tilde{\mu}, \tilde{C} \tag{1.9}
$$

• How to find the most likely explanation of $\mathcal{D}$? Maximum likelihood

$$
[\hat{\mu}_{\text{MLE}}, \hat{C}_{\text{MLE}}] = \arg\max_{\mu \in \mathbb{R}^d, C \in \mathcal{P}(n)} \log \Pr(\mathcal{D}|\mu, C) \tag{1.10}
$$
Learning Gaussians

- Given the data sample $\mathcal{D}$ without knowing $\mu, C$, how likely is the sample under some hypothesized parameters $\tilde{\mu}, \tilde{C}$.

\[
\log \Pr(\mathcal{D}|\tilde{\mu}, \tilde{C}) = \sum_{i=1}^{n} \log \Pr(x_i|\tilde{\mu}, \tilde{C}) \tag{1.8}
\]

\[
\Delta \equiv \log \text{Likelihood of } \mathcal{D} \text{ given } \tilde{\mu}, \tilde{C} \tag{1.9}
\]

- How to find the most likely explanation of $\mathcal{D}$? Maximum likelihood

\[
[\hat{\mu}_{\text{MLE}}, \hat{C}_{\text{MLE}}] = \arg\max_{\mu \in \mathbb{R}^d, C \in \mathcal{P}(n)} \log \Pr(\mathcal{D}|\mu, C) \tag{1.10}
\]

- How to find MLE quantities, again calculus, $\frac{\partial}{\partial \mu} \log \Pr(\mathcal{D}|\mu, C) = 0$ and $\frac{\partial}{\partial C} \log \Pr(\mathcal{D}|\mu, C) = 0$ leads to

\[
\mu_{\text{MLE}} = \frac{1}{n} \sum_{i=1}^{n} x_i \text{ and } C_{\text{MLE}} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_{\text{MLE}})(x_i - \mu_{\text{MLE}})^T \tag{1.11}
\]
 Associations and Associative Memories

Pavlov’s Dog

1. A dish of food
2. A bell ringing
3. The dog salivating
4. The dog salivating while the bell is ringing
Associative memory, auto-associative memory, or hetero-associative memory. In general, associate $x \in \mathcal{X}$ to $y \in \mathcal{Y}$ via $h : \mathcal{X} \rightarrow \mathcal{Y}$. 

Examples: memo memory subsystem (separate address for each $x \in \mathcal{X}$), data structures like hash tables, or red-black trees, etc. Often $\mathcal{X}$, $\mathcal{Y}$ is very large, and we have only a sample associations $D = \{ (x_i, y_i) \}_{i=1}^n$ where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$ where $n \ll |\mathcal{X}|$.

We want to build an associative memory that works even outside of $D$. That is, even for $x /\in \{ x : x = x_i \text{ for some } i \in [n], (x_i, y_i) \in D \}$.

Why? $D$ might not be complete, variation, noise, or possible data corruption not fully captured in $D$. Also, $\mathcal{X}$ might be infinitely large.
Associative memory, auto-associative memory, or hetero-associative memory. In general, associate $x \in \mathcal{X}$ to $y \in \mathcal{Y}$ via $h : \mathcal{X} \rightarrow \mathcal{Y}$.

Examples: memory subsystem (separate address for each $x \in \mathcal{X}$), data structures like hash tables, or red-black trees, etc.
Associations and Associative Memories

- **Associative memory**, auto-associative memory, or hetero-associative memory. In general, associate $x \in \mathcal{X}$ to $y \in \mathcal{Y}$ via $h : \mathcal{X} \rightarrow \mathcal{Y}$.

- Examples: memory subsystem (separate address for each $x \in \mathcal{X}$), data structures like hash tables, or red-black trees, etc.

- Often $\mathcal{X}, \mathcal{Y}$ is very large, and we have only a sample associations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ where $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$ where $n \ll |\mathcal{X}|$. 
Associations and Associative Memories

- **Associative memory**, auto-associative memory, or hetero-associative memory. In general, associate $x \in \mathcal{X}$ to $y \in \mathcal{Y}$ via $h : \mathcal{X} \rightarrow \mathcal{Y}$.

- Examples: memory subsystem (separate address for each $x \in \mathcal{X}$), data structures like hash tables, or red-black trees, etc.

- Often $\mathcal{X}, \mathcal{Y}$ is very large, and we have only a sample associations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$ where $n \ll |\mathcal{X}|$.

- We want to build an associative memory that works even outside of $\mathcal{D}$. That is, even for $x \not\in \{x : x = x_i$ for some $i \in [n], (x_i, y_i) \in \mathcal{D}\}$.
Associations and Associative Memories

- **Associative memory**, auto-associative memory, or hetero-associative memory. In general, associate $x \in \mathcal{X}$ to $y \in \mathcal{Y}$ via $h : \mathcal{X} \rightarrow \mathcal{Y}$.

- Examples: memory subsystem (separate address for each $x \in \mathcal{X}$), data structures like hash tables, or red-black trees, etc.

- Often $\mathcal{X}$, $\mathcal{Y}$ is very large, and we have only a sample associations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$ where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$ where $n \ll |\mathcal{X}|$.

- We want to build an associative memory that works even outside of $\mathcal{D}$. That is, even for $x \not\in \{x : x = x_i \text{ for some } i \in [n], (x_i, y_i) \in \mathcal{D}\}$.

- **Why?** $\mathcal{D}$ might not be complete, variation, noise, or possible data corruption not fully captured in $\mathcal{D}$. Also, $\mathcal{X}$ might be infinitely large.
Machine learning: Write an algorithm that, given large enough $\mathcal{D}$, produces a program $h$ that generalizes (works) well on unseen samples.
Associations and Associative Memories

- Machine learning: Write an algorithm that, given large enough $D$, produces a program $h$ that generalizes (works) well on unseen samples. Respond reasonably to variation, noise, data corruption (be robust).
Associations and Associative Memories

- Machine learning: Write an algorithm that, given large enough $D$, produces a program $h$ that generalizes (works) well on unseen samples. Respond reasonably to variation, noise, data corruption (be robust). Do this computationally as efficiently as possible, and (ideally) understand it mathematically.
Machine learning: Write an algorithm that, given large enough $\mathcal{D}$, produces a program $h$ that generalizes (works) well on unseen samples. Respond reasonably to variation, noise, data corruption (be robust). Do this computationally as efficiently as possible, and (ideally) understand it mathematically.

Boils down to finding a good $h : \mathcal{X} \rightarrow \mathcal{Y}$ that can do the mapping (association). Sometimes we choose some $h \in \mathcal{H}$ where $\mathcal{H}$ is large collection of possible associators. More frequently, $h$ is parameterized via some parameters $\theta$ and we find a good $\theta$ leading to $h_\theta$. 
Associations and Associative Memories

- Machine learning: Write an algorithm that, given large enough $D$, produces a program $h$ that generalizes (works) well on unseen samples. Respond reasonably to variation, noise, data corruption (be robust). Do this computationally as efficiently as possible, and (ideally) understand it mathematically.

- Boils down to finding a good $h : \mathcal{X} \rightarrow \mathcal{Y}$ that can do the mapping (association). Sometimes we choose some $h \in \mathcal{H}$ where $\mathcal{H}$ is large collection of possible associators. More frequently, $h$ is parameterized via some parameters $\theta$ and we find a good $\theta$ leading to $h_\theta$.

- Many ways to do this, depends on nature of $\mathcal{X}, \mathcal{Y}$, how big the data is (number of samples $n$), and available resources (compute, core machine memory/RAM, storage/disk, communication (latency/bandwidth), time, money, energy usage).
Associations and Associative Memories

- Machine learning: Write an algorithm that, given large enough \( \mathcal{D} \), produces a program \( h \) that generalizes (works) well on unseen samples. Respond reasonably to variation, noise, data corruption (be robust). Do this computationally as efficiently as possible, and (ideally) understand it mathematically.

- Boils down to finding a good \( h : \mathcal{X} \rightarrow \mathcal{Y} \) that can do the mapping (association). Sometimes we choose some \( h \in \mathcal{H} \) where \( \mathcal{H} \) is large collection of possible associators. More frequently, \( h \) is parameterized via some parameters \( \theta \) and we find a good \( \theta \) leading to \( h_\theta \).

- Many ways to do this, depends on nature of \( \mathcal{X}, \mathcal{Y} \), how big the data is (number of samples \( n \)), and available resources (compute, core machine memory/RAM, storage/disk, communication (latency/bandwidth), time, money, energy usage).

- Often, \( x \in \mathbb{R}^m \) is an \( m \)-dimensional vector of features. In general, \( x \) is known as a feature vector.
1D Example

- Associating living area with price, or learn to map from living area to home price.

<table>
<thead>
<tr>
<th>Living area (feet^2)</th>
<th>Price (1000$s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2104</td>
<td>400</td>
</tr>
<tr>
<td>1600</td>
<td>330</td>
</tr>
<tr>
<td>2400</td>
<td>369</td>
</tr>
<tr>
<td>1416</td>
<td>232</td>
</tr>
<tr>
<td>3000</td>
<td>540</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

These and next few examples from Andrew Ng's class: http://cs229.stanford.edu/syllabus.html
1D Example

Associating living area with price, or learn to map from living area to home price.

Can plot this data

<table>
<thead>
<tr>
<th>Living area (feet²)</th>
<th>Price (1000$s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2104</td>
<td>400</td>
</tr>
<tr>
<td>1600</td>
<td>330</td>
</tr>
<tr>
<td>2400</td>
<td>369</td>
</tr>
<tr>
<td>1416</td>
<td>232</td>
</tr>
<tr>
<td>3000</td>
<td>540</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

These and next few examples from Andrew Ng’s class: http://cs229.stanford.edu/syllabus.html
1D Example

**Associating living area with price, or learn to map from living area to home price.**

- Can plot this data

- Here $\mathcal{X} = \mathcal{Y} = \mathbb{R}$.

These and next few examples from Andrew Ng’s class: http://cs229.stanford.edu/syllabus.html
Higher Dimensions

- Can use other inputs as well.

<table>
<thead>
<tr>
<th>Living area (feet²)</th>
<th>#bedrooms</th>
<th>Price (1000$s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2104</td>
<td>3</td>
<td>400</td>
</tr>
<tr>
<td>1600</td>
<td>3</td>
<td>330</td>
</tr>
<tr>
<td>2400</td>
<td>3</td>
<td>369</td>
</tr>
<tr>
<td>1416</td>
<td>2</td>
<td>232</td>
</tr>
<tr>
<td>3000</td>
<td>4</td>
<td>540</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Higher Dimensions

- Can use other inputs as well.

- Tradeoff: The good: more inputs, more information, more potential for accurate association. The bad: higher dimensional space, need for much larger $n$, curse of dimensionality.
Statistical parameter estimation

- Training data $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $(x^{(i)}, y^{(i)}) \sim p(x, y)$ are drawn from some distribution, $x^{(i)} \in \mathbb{R}^m$ and $y^{(i)} \in \mathbb{R}$.
Statistical parameter estimation

- Training data \( \mathcal{D} = \{ (x^{(i)}, y^{(i)}) \}_{i=1}^{n} \) where \( (x^{(i)}, y^{(i)}) \sim p(x, y) \) are drawn from some distribution, \( x^{(i)} \in \mathbb{R}^m \) and \( y^{(i)} \in \mathbb{R} \).
- \( x^{(i)} \) is \( m \)-dimensional column vector of features, \( y^{(i)} \) is scalar.
Statistical parameter estimation

- Training data $\mathcal{D} = \left\{ (x^{(i)}, y^{(i)}) \right\}_{i=1}^{n}$ where $(x^{(i)}, y^{(i)}) \sim p(x, y)$ are drawn from some distribution, $x^{(i)} \in \mathbb{R}^m$ and $y^{(i)} \in \mathbb{R}$.
- $x^{(i)}$ is $m$-dimensional column vector of features, $y^{(i)}$ is scalar.
- Goal: find $h_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ with minimum error, where

$$\text{Error}_i = e_i = h_\theta(x^{(i)}) - y^{(i)}$$

$$E[e^2] = E_{p(x,y)}[(h_\theta(x) - y)^2] = \int p(x, y)(h_\theta(x) - y)^2 dxdy$$

$$= \int p(x) \int (h_\theta(x) - y)^2 p(y|x) dydx$$

and $\theta \in \mathbb{R}^m$ is a parameter vector, $\theta = (\theta_1, \theta_2, \ldots, \theta_m)$, $\theta_i \in \mathbb{R}$. 
Statistical parameter estimation

- Training data \( \mathcal{D} = \{ (x^{(i)}, y^{(i)}) \}_{i=1}^n \) where \((x^{(i)}, y^{(i)}) \sim p(x, y)\) are drawn from some distribution, \(x^{(i)} \in \mathbb{R}^m\) and \(y^{(i)} \in \mathbb{R}\).
- \(x^{(i)}\) is \(m\)-dimensional column vector of features, \(y^{(i)}\) is scalar.
- Goal: find \(h_\theta : \mathcal{X} \rightarrow \mathcal{Y}\) with minimum error, where

\[
\text{Error}_i = e_i = h_\theta(x^{(i)}) - y^{(i)}
\]  

(1.12)

\[
E[e^2] = E_{p(x,y)}[(h_\theta(x) - y)^2] = \int p(x, y)(h_\theta(x) - y)^2 dx dy
\]  

(1.13)

\[
= \int p(x) \int (h_\theta(x) - y)^2 p(y|x) dy dx
\]  

(1.14)

and \(\theta \in \mathbb{R}^m\) is a parameter vector, \(\theta = (\theta_1, \theta_2, \ldots, \theta_m), \theta_i \in \mathbb{R}\).

- Taking derivatives and setting to zero, we get best solution:

\[
h_\theta(x) = \int y p(y|x) dy = E[Y|x] = \text{best association.}
\]  

(1.15)
Statistical parameter estimation

- Training data $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ where $(x^{(i)}, y^{(i)}) \sim p(x, y)$ are drawn from some distribution, $x^{(i)} \in \mathbb{R}^m$ and $y^{(i)} \in \mathbb{R}$.
- $x^{(i)}$ is $m$-dimensional column vector of features, $y^{(i)}$ is scalar.
- Goal: find $h_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ with minimum error, where

\[
\text{Error}_i = e_i = h_\theta(x^{(i)}) - y^{(i)}
\]  

\[
E[e^2] = E_{p(x,y)}[(h_\theta(x) - y)^2] = \int p(x, y)(h_\theta(x) - y)^2 dxdy
\]  

\[
= \int p(x) \int (h_\theta(x) - y)^2 p(y|x) dy dx
\]

and $\theta \in \mathbb{R}^m$ is a parameter vector, $\theta = (\theta_1, \theta_2, \ldots, \theta_m)$, $\theta_i \in \mathbb{R}$.
- Taking derivatives and setting to zero, we get best solution:

\[
h_\theta(x) = \int y p(y|x) dy = E[Y|x] = \text{best association.}
\]  

- This assumes we have the distribution $p$ and also the resources to compute $E[Y|x]$. 

Linear estimator: Linear regression

- We can find best solution under a linear model, where $h_\theta(x) = \theta^T x$, where $\theta$ is an $m \times 1$ column vector of “regression” coefficients, in which case $\hat{y} = \theta^T x$
Linear estimator: Linear regression

- We can find best solution under a linear model, where $h_\theta(x) = \theta^T x$, where $\theta$ is an $m \times 1$ column vector of “regression” coefficients, in which case $\hat{y} = \theta^T x$

- In general, assume $h_\theta(x) \triangleq \theta^T x$ is parameterized by parameters $\theta$ so

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$$  \hspace{1cm} (1.16)
Linear estimator: Linear regression

- We can find best solution under a linear model, where \( h_\theta(x) = \theta^T x \), where \( \theta \) is an \( m \times 1 \) column vector of “regression” coefficients, in which case \( \hat{y} = \theta^T x \)
- In general, assume \( h_\theta(x) \triangleq \theta^T x \) is parameterized by parameters \( \theta \) so

\[
J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2
\] (1.16)

- Bias: sometimes we fix \( x_m^{(i)} = 1 \) for all \( i \) so that \( \theta_m \) is a bias, or offset in the linear (or affine) model.
Linear estimator: Linear regression

- We can find best solution under a linear model, where \( h_\theta(x) = \theta^T x \), where \( \theta \) is an \( m \times 1 \) column vector of “regression” coefficients, in which case \( \hat{y} = \theta^T x \).

- In general, assume \( h_\theta(x) \triangleq \theta^T x \) is parameterized by parameters \( \theta \) so

\[
J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2
\]

(1.16)

- **Bias:** sometimes we fix \( x_m^{(i)} = 1 \) for all \( i \) so that \( \theta_m \) is a bias, or offset in the linear (or affine) model.

- **Taking derivative of error objective** \( J(\theta) \) w.r.t. \( \theta \) and set to zero gets:

\[
\frac{\partial J}{\partial \theta} = \frac{2}{n} \sum_{n=1}^{n} (h_\theta(x^{(i)}) - y^{(i)}) \frac{\partial h_\theta(x^{(i)})}{\partial \theta} = 0
\]

(1.17)
Linear estimator: Linear regression

- We can find best solution under a linear model, where \( h_\theta(x) = \theta^T x \), where \( \theta \) is an \( m \times 1 \) column vector of “regression” coefficients, in which case \( \hat{y} = \theta^T x \)

- In general, assume \( h_\theta(x) \triangleq \theta^T x \) is parameterized by parameters \( \theta \) so

\[
J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1.16}
\]

- Bias: sometimes we fix \( x_m^{(i)} = 1 \) for all \( i \) so that \( \theta_m \) is a bias, or offset in the linear (or affine) model.

- Taking derivative of error objective \( J(\theta) \) w.r.t. \( \theta \) and set to zero gets:

\[
\frac{\partial J}{\partial \theta} = \frac{2}{n} \sum_{n=1}^{n} (h_\theta(x^{(i)}) - y^{(i)}) \frac{\partial h_\theta(x^{(i)})}{\partial \theta} = 0 \tag{1.17}
\]

- Linear \( h_\theta(x) = x^T \theta \) assumption, yields \( \frac{\partial h_\theta(x^{(i)})}{\partial \theta} = x^{(i)} \).
Linear regression: feature vectors and transformations

- Linear model, $h_\theta(x) = \theta^T x$, linear both in $\theta$ and in $x \in \mathbb{R}^m$. 

---

Prof. Jeff Bilmes  
EE511/Spring 2020/Adv. Intro ML - Week 1 - March 30th, 2020  
F36/49 (pg.93/131)
Linear regression: feature vectors and transformations

- Linear model, \( h_\theta(x) = \theta^T x \), linear both in \( \theta \) and in \( x \in \mathbb{R}^m \).
- \( x \in \mathbb{R}^m \) is a feature vector but it might be derived from some other (more raw) information sources, say \( z \in \mathbb{R}^{m'} \) for some \( m' \neq m \).
Linear regression: feature vectors and transformations

- Linear model, $h_\theta(x) = \theta^\top x$, linear both in $\theta$ and in $x \in \mathbb{R}^m$.
- $x \in \mathbb{R}^m$ is a feature vector but it might be derived from some other (more raw) information sources, say $z \in \mathbb{R}^{m'}$ for some $m' \neq m$.
- Example, $z$ might be raw sensor values, pixel values, or audio sample values (anything close to or at the sensor), and $x$ might be some deterministic (i.e., non-random) derived processing done on $z$. 
Linear regression: feature vectors and transformations

- Linear model, \( h_\theta(x) = \theta^\top x \), linear both in \( \theta \) and in \( x \in \mathbb{R}^m \).
- \( x \in \mathbb{R}^m \) is a feature vector but it might be derived from some other (more raw) information sources, say \( z \in \mathbb{R}^{m'} \) for some \( m' \neq m \).
- Example, \( z \) might be raw sensor values, pixel values, or audio sample values (anything close to or at the sensor), and \( x \) might be some deterministic (i.e., non-random) derived processing done on \( z \).
- Simple example: \( m' = 2 \), \( z = (z_1, z_2) \), \( x = (x_1, x_2, \ldots, x_m) \), where 
  \[ 
  x_1 = z_1, \quad x_2 = z_2, \quad x_3 = z_1^2, \quad x_4 = z_2^2, \quad x_5 = z_1 z_2, \quad x_6 = z_1^2 z_2, \quad x_7 = z_1 z_2^2, \\
  x_8 = z_1^3, \quad x_9 = z_1^2 z_2^2, \quad \text{etc.} 
  \]
Linear regression: feature vectors and transformations

- Linear model, \( h_\theta(x) = \theta^\top x \), linear both in \( \theta \) and in \( x \in \mathbb{R}^m \).
- \( x \in \mathbb{R}^m \) is a feature vector but it might be derived from some other (more raw) information sources, say \( z \in \mathbb{R}^{m'} \) for some \( m' \neq m \).
- Example, \( z \) might be raw sensor values, pixel values, or audio sample values (anything close to or at the sensor), and \( x \) might be some deterministic (i.e., non-random) derived processing done on \( z \).
- Simple example: \( m' = 2 \), \( z = (z_1, z_2) \), \( x = (x_1, x_2, \ldots, x_m) \), where \( x_1 = z_1 \), \( x_2 = z_2 \), \( x_3 = z_1^2 \), \( x_4 = z_2^2 \), \( x_5 = z_1 z_2 \), \( x_6 = z_1^2 z_2 \), \( x_7 = z_1 z_2^2 \), \( x_8 = z_1^3 \), \( x_9 = z_1^3 z_2^2 \), etc.
- Whatever the transformation, lets say that \( x = \phi(z) \) for some fixed, non-learnt, transformation function \( \phi : \mathbb{R}^{m'} \to \mathbb{R}^m \). Sometimes this is known as feature extraction. Sometimes learnt representation learning.
Linear regression: feature vectors and transformations

- Linear model, $h_\theta(x) = \theta^\top x$, linear both in $\theta$ and in $x \in \mathbb{R}^m$.
- $x \in \mathbb{R}^m$ is a feature vector but it might be derived from some other (more raw) information sources, say $z \in \mathbb{R}^{m'}$ for some $m' \neq m$.
- Example, $z$ might be raw sensor values, pixel values, or audio sample values (anything close to or at the sensor), and $x$ might be some deterministic (i.e., non-random) derived processing done on $z$.
- Simple example: $m' = 2$, $z = (z_1, z_2)$, $x = (x_1, x_2, \ldots, x_m)$, where $x_1 = z_1$, $x_2 = z_2$, $x_3 = z_1^2$, $x_4 = z_2^2$, $x_5 = z_1 z_2$, $x_6 = z_1^2 z_2$, $x_7 = z_1 z_2^2$, $x_8 = z_1^3$, $x_9 = z_1^3 z_2^2$, etc.
- Whatever the transformation, lets say that $x = \phi(z)$ for some fixed, non-learnt, transformation function $\phi : \mathbb{R}^{m'} \to \mathbb{R}^m$. Sometimes this is known as feature extraction. Sometimes learnt representation learning.
- Thus, while $h_\theta(x) = \theta^\top x = \theta^\top \phi(z)$ is linear in $\theta$ and $x$, it need not be linear (and is likely very non-linear) in $z$. 
Introduction
Uncertainty & Probability
Coins
Gaussians
AMs
Supervised
Linear Regression
LLS, Batch vs. Online, SGD
Fit

Linear regression: feature vectors and transformations

- Linear model, $h_\theta(x) = \theta^T x$, linear both in $\theta$ and in $x \in \mathbb{R}^m$.
- $x \in \mathbb{R}^m$ is a feature vector but it might be derived from some other (more raw) information sources, say $z \in \mathbb{R}^{m'}$ for some $m' \neq m$.
- Example, $z$ might be raw sensor values, pixel values, or audio sample values (anything close to or at the sensor), and $x$ might be some deterministic (i.e., non-random) derived processing done on $z$.
- Simple example: $m' = 2$, $z = (z_1, z_2)$, $x = (x_1, x_2, \ldots, x_m)$, where $x_1 = z_1$, $x_2 = z_2$, $x_3 = z_1^2$, $x_4 = z_2^2$, $x_5 = z_1 z_2$, $x_6 = z_1^2 z_2$, $x_7 = z_1 z_2^2$, $x_8 = z_1^3$, $x_9 = z_1^3 z_2^2$, etc.
- Whatever the transformation, lets say that $x = \phi(z)$ for some fixed, non-learnt, transformation function $\phi : \mathbb{R}^{m'} \rightarrow \mathbb{R}^m$. Sometimes this is known as feature extraction. Sometimes learnt representation learning.
- Thus, while $h_\theta(x) = \theta^T x = \theta^T \phi(z)$ is linear in $\theta$ and $x$, it need not be linear (and is likely very non-linear) in $z$.
- Offers linear models much more expressiveness, ability, complexity (etc.), but also a risk for overfitting if too many features.
Linear Least Squares

- This gives objective to be minimized (smallest, or least of the sum of squares of the errors).

\[ \frac{\partial J(\theta)}{\partial \theta} = \frac{2}{n} \sum_{n=1}^{n} (x^{(i)^T} \theta - y^{(i)}) x^{(i)} = 0 \]  

(1.18)
Linear Least Squares

- This gives objective to be minimized (smallest, or least of the sum of squares of the errors).

\[
\frac{\partial J(\theta)}{\partial \theta} = 2 \sum_{n=1}^{n} (x(i)^T \theta - y(i)) x(i) = 0
\]  

(1.18)

- We simplify this a bit by defining matrices associated with these quantities. First define a \( n \times m \) **design matrix** \( X \) and length-\( n \) column vector \( \vec{y} \)

\[
X = \begin{pmatrix}
    x^{(1)^T} & \vdots & x^{(n)^T}
\end{pmatrix}, \quad \text{and} \quad \vec{y} = \begin{pmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n
\end{pmatrix}
\]  

(1.19)
Linear Least Squares

- This gives objective to be minimized (smallest, or least of the sum of squares of the errors).

\[
\frac{\partial J(\theta)}{\partial \theta} = 2 \sum_{n=1}^{n} (x^{(i)^T} \theta - y^{(i)}) x^{(i)} = 0
\]  (1.18)

- We simplify this a bit by defining matrices associated with these quantities. First define a \( n \times m \) design matrix \( X \) and length-\( n \) column vector \( \vec{y} \)

\[
X = \begin{pmatrix}
- & x^{(1)^T} & - \\
- & x^{(2)^T} & - \\
\vdots & \vdots & \vdots \\
- & x^{(n)^T} & -
\end{pmatrix}, \text{ and } \vec{y} = \begin{pmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n
\end{pmatrix}
\]  (1.19)

- Objective Equation (1.18), equivalent matrix-vector form:

\[
J(\theta) = \frac{1}{2} (X \theta - \vec{y})^T (X \theta - \vec{y})
\]  (1.20)
Normal Equations

With this, we get the “normal equations”

$$\nabla_\theta J(\theta) = X^\top (X\theta - \vec{y}) = \vec{0}$$  \hspace{1cm} (1.21)

i.e., modeling $\vec{y}$ to be in column space of matrix $X$ (linear combinations of columns of $X$), when $\vec{y}$ is being approximated by $X\theta$. 
With this, we get the “normal equations”

$$\nabla_\theta J(\theta) = X^\top (X\theta - \vec{y}) = \vec{0}$$

(1.21)

i.e., modeling $\vec{y}$ to be in column space of matrix $X$ (linear combinations of columns of $X$), when $\vec{y}$ is being approximated by $X\theta$.

Called normal equations because column space of $X$ is orthogonal to the residual error $E = (\vec{y} - X\theta)$, giving solution $\theta = \tilde{\theta}$ as shown.

what is to be approximated

$\vec{y}$

$\vec{y} - X\tilde{\theta}$

column space of $X$

space of possible approximations, $\theta \in \mathbb{R}^m$

actual approximation, closest point to $\vec{y}$

error

what is to be approximated

$\{\vec{y} : y = X\theta, \theta \in \mathbb{R}^m\}$

Normal Equations
Normal Equations

- With this, we get the “normal equations”

\[
\nabla_\theta J(\theta) = X^\top (X\theta - \vec{y}) = \vec{0}
\]

i.e., modeling \( \vec{y} \) to be in column space of matrix \( X \) (linear combinations of columns of \( X \)), when \( \vec{y} \) is being approximated by \( X\theta \).

- Called normal equations because column space of \( X \) is orthogonal to the residual error \( E = (\vec{y} - X\theta) \), giving solution \( \theta = \tilde{\theta} \) as shown.

If \( X^\top X \) invertible (typical if \( n \gg m \)), solution has form:

\[
\tilde{\theta} = (X^\top X)^{-1} X^\top \vec{y}
\]

where \( (X^\top X)^{-1} X^\top \) is known as the Moore-Penrose pseudo-inverse of matrix \( X \).
Batch vs. Online

- Linear least squares is a batch algorithm, it uses all of the data at the same time and requires random access (for the matrix inversion).
Batch vs. Online

- Linear least squares is a **batch** algorithm, it uses all of the data at the same time and requires random access (for the matrix inversion).
- It “learns” in one step, via an analytically solving a matrix equation.
Batch vs. Online

- Linear least squares is a **batch** algorithm, it uses all of the data at the same time and requires random access (for the matrix inversion).
- It "learns" in one step, via an analytically solving a matrix equation.
- For many types of models, there is no analytical solution to find the solution. Also, we prefer an **online** or **streaming** method that adjusts parameters as the data comes in, it does not require all data in memory simultaneously nor does it require random access.
Batch vs. Online

- Linear least squares is a **batch** algorithm, it uses all of the data at the same time and requires random access (for the matrix inversion).

- It “learns” in one step, via an analytically solving a matrix equation.

- For many types of models, there is no analytical solution to find the solution. Also, we prefer an **online** or **streaming** method that adjusts parameters as the data comes in, it does not require all data in memory simultaneously nor does it require random access.

- The approach taken is to move parameters $\theta$ in the direction of the gradient $\nabla_{\theta} J(\theta)$ a certain amount and to do that repeatedly.
Batch vs. Online

- Linear least squares is a **batch** algorithm, it uses all of the data at the same time and requires random access (for the matrix inversion).
- It “learns” in one step, via an analytically solving a matrix equation.
- For many types of models, there is no analytical solution to find the solution. Also, we prefer an **online** or streaming method that adjusts parameters as the data comes in, it does not require all data in memory simultaneously nor does it require random access.
- The approach taken is to move parameters $\theta$ in the direction of the gradient $\nabla_\theta J(\theta)$ a certain amount and to do that repeatedly.
- Gradient descent can itself both be batch and online.
Batch Gradient Descent

- Start with initial estimate $\theta \in \mathbb{R}^m$ and update it, for all $j \in [m]$ via

$$
\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)
$$

(1.22)

where

$$
\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2}(h_\theta(x) - y)^2
$$

(1.23)

$$
= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (h_\theta(x) - y)
$$

(1.24)

$$
= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} \left( \sum_j \theta_j x_j - y \right)
$$

(1.25)

$$
= (h_\theta(x) - y) x_j
$$

(1.26)
Start with initial estimate $\theta \in \mathbb{R}^m$ and update it, for all $j \in [m]$ via

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{1.22}$$

where

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2$$

$$= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \tag{1.23}$$

$$= (h_\theta(x) - y) \frac{\partial}{\partial \theta_j} \left( \sum_j \theta_j x_j - y \right) \tag{1.24}$$

$$= (h_\theta(x) - y) x_j \tag{1.25}$$

This leads to update rule, for all $j$:

$$\theta_j \leftarrow \theta_j + \alpha (y - h_\theta(x)) x_j \tag{1.27}$$
Batch Gradient Descent

- Gradient updates for all elements of $\theta$ at the same time and for sample pair $(x^{(i)}, y^{(i)})$

$$\theta \leftarrow \theta + \alpha(y^{(i)} - h_\theta(x^{(i)}))x^{(i)} = \theta + \alpha(y^{(i)} - \theta^T x^{(i)})x^{(i)} \quad (1.28)$$

move $\theta$ in the direction of $x^{(i)}$ weighted by $\alpha(y^{(i)} - h_\theta(x^{(i)})) \in \mathbb{R}$, $\alpha$ times the error.
Batch Gradient Descent

- Gradient updates for all elements of $\theta$ at the same time and for sample pair $(x^{(i)}, y^{(i)})$

$$\theta \leftarrow \theta + \alpha(y^{(i)} - h_{\theta}(x^{(i)}))x^{(i)} = \theta + \alpha(y^{(i)} - \theta^T x^{(i)})x^{(i)}$$

(1.28)

move $\theta$ in the direction of $x^{(i)}$ weighted by $\alpha(y^{(i)} - h_{\theta}(x^{(i)})) \in \mathbb{R}$, $\alpha$ times the error.

- Called **LMS** (least mean squares) update rule, also called **Widrow-Hoff** (early NN folks) learning rule.
Batch Gradient Descent

- Gradient updates for all elements of $\theta$ at the same time and for sample pair $(x^{(i)}, y^{(i)})$

$$\theta \leftarrow \theta + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x^{(i)} = \theta + \alpha (y^{(i)} - \theta^T x^{(i)}) x^{(i)} \quad (1.28)$$

move $\theta$ in the direction of $x^{(i)}$ weighted by $\alpha (y^{(i)} - h_{\theta}(x^{(i)})) \in \mathbb{R}$, $\alpha$ times the error.

- Called LMS (least mean squares) update rule, also called Widrow-Hoff (early NN folks) learning rule.

- **Batch Gradient Descent**

**Algorithm 3:** Batch Gradient descent learning

**Input:** Training data $\mathcal{D}$, learning rate $\alpha$, initial parameter estimate $\theta$

**Output:** Learnt model parameters $\theta$

1. for $t = 1, \ldots, T$ do
2. \[ \theta \leftarrow \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - h_{\theta}(x^{(i)})) x^{(i)} \]

Return: the final parameters $\theta$
Batch training results (left) and resulting fit model (right).
More visualization: Batch Gradient Descent
Stochastic Gradient Descent (SGD)

**Algorithm 4:** Stochastic gradient descent (SGD) learning

**Input:** Training data \( D \), learning rate \( \alpha \), initial parameter estimate \( \theta \)

**Output:** Learnt model parameters \( \theta \)

1. for \( t = 1, \cdots, T \) do
2.     for \( i = 1, \cdots, n \) do
3.         \( \theta \leftarrow \theta + \alpha(y^{(i)} - h_\theta(x^{(i)}))x^{(i)} \)

**Return:** the final parameters \( \theta \)

Optimization folks (e.g., Bertsekas) call this incremental gradient methods. It is stochastic if we randomize (with or without replacement) the order of the data items.
More visualization: Stochastic Gradient Descent
Let $V = \{1, 2, \ldots, n\}$ be the index set of training points and let $V = \{V_1, V_2, \ldots, V_k\}$ be a partition, $V_\ell \subseteq V$ and $V_\ell \cap V_p = \emptyset$ when $\ell \neq p$. Normally we want $|V_\ell| \approx |V_p|$ for all $\ell, p$. 

Mini-Batch Gradient Descent

Algorithm 5:
Minibatch stochastic gradient descent learning

Input: $D$, training data point set, learning rate $\alpha$, initial parameter estimate $\theta$

Output: Learnt model parameters $\theta$

1. for $t = 1, \ldots, T$
2. for $\ell = 1, \ldots, k$
3. $\theta \leftarrow \theta + \alpha \sum_{i \in V_\ell} (y(i) - h_\theta(x(i))) x(i)$

Return: the final parameters $\theta$

Objective as sum of batch errors, of the form

$$J(\theta) = \frac{1}{k} \sum_{\ell=1}^{k} \frac{1}{|V_\ell|} \sum_{i \in V_\ell} (h_\theta(x(i)) - y(i))^2$$ (1.29)
Mini-Batch Gradient Descent

- Let $V = \{1, 2, \ldots, n\}$ be the index set of training points and let $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ be a partition, $V_\ell \subseteq V$ and $V_\ell \cap V_p = \emptyset$ when $\ell \neq p$. Normally we want $|V_\ell| \approx |V_p|$ for all $\ell, p$.
- Each $V_\ell$ is a mini-batch (or bunch) of data points, leading to:

\begin{algorithm}
\textbf{Algorithm 6:} Minibatch stochastic gradient descent learning
\begin{algorithmic}
\State \textbf{Input :} Training data $\mathcal{D}$, learning rate $\alpha$, initial parameter estimate $\theta$
\State \textbf{Output:} Learnt model parameters $\theta$
\For{$t = 1, \cdots, T$}
\For{$\ell = 1, \cdots, k$}
\State $\theta \leftarrow \theta + \alpha \sum_{i \in V_\ell} (y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$
\EndFor
\EndFor
\State \Return the final parameters $\theta$
\end{algorithmic}
\end{algorithm}
Let $V = \{1, 2, \ldots, n\}$ be the index set of training points and let $\mathcal{V} = \{V_1, V_2, \ldots, V_k\}$ be a partition, $V_\ell \subseteq V$ and $V_\ell \cap V_p = \emptyset$ when $\ell \neq p$. Normally we want $|V_\ell| \approx |V_p|$ for all $\ell, p$.

Each $V_\ell$ is a mini-batch (or bunch) of data points, leading to:

### Algorithm 7: Minibatch stochastic gradient descent learning

**Input**: Training data $\mathcal{D}$, learning rate $\alpha$, initial parameter estimate $\theta$

**Output**: Learnt model parameters $\theta$

1. for $t = 1, \cdots, T$ do
2.  for $\ell = 1, \cdots, k$ do
3.    $\theta \leftarrow \theta + \alpha \sum_{i \in V_\ell} (y^{(i)} - h_\theta(x^{(i)})) x^{(i)}$

**Return**: the final parameters $\theta$

Objective as sum of batch errors, of the form

$$J(\theta) = \frac{1}{k} \sum_{\ell=1}^{k} \frac{1}{n} |V_\ell| \left( \frac{1}{|V_\ell|} \sum_{i \in V_\ell} (h_\theta(x^{(i)}) - y^{(i)})^2 \right)$$  (1.29)
Normal Equations and Gaussians

- Assume $y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$ where

$$\epsilon \sim \Pr(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$  \hspace{1cm} (1.30)

or that

$$p(y^{(i)} | x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$  \hspace{1cm} (1.31)
Normal Equations and Gaussians

- Assume $y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$ where

$$
\epsilon \sim \text{Pr}(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)
$$

or that

$$
p(y^{(i)} | x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)
$$

- Log likelihood of data given parameters

$$
\text{Likelihood}(\theta) = \log \prod_{i=1}^{n} p(y^{(i)} | x^{(i)}; \theta)
$$
Normal Equations and Gaussians

- Assume $y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$ where

$$
\epsilon \sim \Pr(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)
$$

(1.30)

or that

$$
p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)
$$

(1.31)

- Log likelihood of data given parameters

$$
\text{Likelihood}(\theta) = \log \prod_{i=1}^{n} p(y^{(i)}|x^{(i)}; \theta)
$$

(1.32)

- Taking derivatives and setting them $= 0$ yields exactly same normal equations we saw earlier for solving linear least squares. Does not use $\sigma$. 

Prof. Jeff Bilmes
Fit a model with various input features, values of powers of $x$, goal is to predict $y$ based on xy-pair samples $D = \{(x^{(i)}, y^{(i)})\}_i$. 

- Left is underfitting. Right is overfitting.
Underfitting vs. Overfitting

- Fit a model with various input features, values of powers of \( x \), goal is to predict \( y \) based on \( xy \)-pair samples \( \mathcal{D} = \{(x^{(i)}, y^{(i)})\}_i \).

- Fit models: left \( y = \theta_0 + \theta_1 x \); middle \( y = \theta_0 + \theta_1 x + \theta_2 x^2 \); right \( y = \sum_{j=0}^{5} \theta_j x^j \).
Fit a model with various input features, values of powers of \( x \), goal is to predict \( y \) based on xy-pair samples \( \mathcal{D} = \{(x^{(i)}, y^{(i)})\}_i \).

- **Fit models:** left \( y = \theta_0 + \theta_1 x \); middle \( y = \theta_0 + \theta_1 x + \theta_2 x^2 \); right \( y = \sum_{j=0}^5 \theta_j x^j \).
- **Left is underfitting.** Right is **overfitting.**
We say that a hypothesis overfits the training examples if some other hypothesis that fits the training examples less well actually performs better over the entire distribution of instances (i.e., including instances beyond the training set).
Overfitting definition (T. Mitchell)

- We say that a hypothesis overfits the training examples if some other hypothesis that fits the training examples less well actually performs better over the entire distribution of instances (i.e., including instances beyond the training set).

**Definition 1.10.1 (overfitting)**

Given a hypothesis space $\mathcal{H}$, a hypothesis $h \in \mathcal{H}$ is said to overfit the training data if there exists some alternative hypothesis $h' \in \mathcal{H}$, such that $h$ has smaller error than $h'$ over the training examples, but $h'$ has a smaller overall error than $h$ over the entire distribution (or data set) of instances.
Overfitting definition (T. Mitchell)

- We say that a hypothesis overfits the training examples if some other hypothesis that fits the training examples less well actually performs better over the entire distribution of instances (i.e., including instances beyond the training set).

Definition 1.10.1 (overfitting)

Given a hypothesis space \( \mathcal{H} \), a hypothesis \( h \in \mathcal{H} \) is said to overfit the training data if there exists some alternative hypothesis \( h' \in \mathcal{H} \), such that \( h \) has smaller error than \( h' \) over the training examples, but \( h' \) has a smaller overall error than \( h \) over the entire distribution (or data set) of instances.

- We’ll visit this topic again when we discuss bias/variance, but first lets discuss a few more models.