Midterm Review
CS 6375: Machine Learning

Vibhav Gogate
The University of Texas at Dallas
Machine Learning

Supervised Learning
- Parametric
  - Gaussians
    - Learned in closed form
  - Linear Functions
    1. Learned in closed form
    2. Using gradient descent
- Non-parametric
  - Y Continuous
- Y Discrete
  - Reinforcement Learning
  - Nearest Neighbor methods
  - Locally weighted regression

Unsupervised Learning
- Parametric
- Non-parametric
  - Non-probabilistic
- Linear: perceptron gradient descent
- Nonlinear: neural net: backprop
- Support vector machines

Decision Trees
- Greedy search; pruning
- Probability of class | features
  1. Learn P(Y), P(X|Y); apply Bayes
  2. Learn P(Y|X) w/ gradient descent

Support vector machines
Supervised Learning

- **Given:** Training examples \( (x, f(x)) \) for some unknown function \( f \).
- **Find:** A good approximation to \( f \).

Example Applications

- **Credit risk assessment**
  \( x \): Properties of customer and proposed purchase.
  \( f(x) \): Approve purchase or not.

- **Disease diagnosis**
  \( x \): Properties of patient (symptoms, lab tests)
  \( f(x) \): Disease (or maybe, recommended therapy)

- **Face recognition**
  \( x \): Bitmap picture of person’s face
  \( f(x) \): Name of the person.

- **Automatic Steering**
  \( x \): Bitmap picture of road surface in front of car.
  \( f(x) \): Degrees to turn the steering wheel.
Key Issues in Machine Learning

- **What are good hypothesis spaces?**
  Which spaces have been useful in practical applications and why?

- **What algorithms can work with these spaces?**
  Are there general design principles for machine learning algorithms?

- **How can we optimize accuracy on future data points?**
  This is sometimes called the “problem of overfitting”.

- **How can we have confidence in the results?**
  How much training data is required to find accurate hypotheses? (the *statistical question*)

- **Are some learning problems computationally intractable?**
  (the *computational question*)

- **How can we formulate application problems as machine learning problems?** (the *engineering question*)
Key Perspective on Learning

- Learning as Optimization
  - Closed form
  - Greedy search
  - Gradient ascent

- Loss Function
  - Error + regularization
Decision Tree Hypothesis Space

- **Internal nodes** test the value of particular features $x_j$ and branch according to the results of the test.

- **Leaf nodes** specify the class $h(x)$.

![Decision Tree Diagram]

Suppose the features are **Outlook** ($x_1$), **Temperature** ($x_2$), **Humidity** ($x_3$), and **Wind** ($x_4$). Then the feature vector $\mathbf{x} = (\text{Sunny, Hot, High, Strong})$ will be classified as No. The **Temperature** feature is irrelevant.
Decision Tree Decision Boundaries

Decision trees divide the feature space into axis-parallel rectangles, and label each rectangle with one of the $K$ classes.
What you should know in Decision Tree Learning?

• Heuristics for selecting the next attribute
  – Information gain, one-step look ahead, gain ratio
  – What makes the heuristic good?
  – What are its cons?
  – Complexity analysis
  – Sample exam question: if I tweak the selection heuristic, how will that change the complexity and quality?

• What kind of functions it can learn.
• Overfitting and Pruning
• Handling missing data
• Handling continuous attributes
• Noise
• Small number of examples associated with each leaf
  • What if only one example is associated with a leaf. Can you believe it?
• Coincidental regularities
Probability Theory

• Be able to apply and understand
  – Axioms of probability
  – Distribution vs density
  – Conditional probability
  – Sum-rule, chain-rule
  – Bayes rule

• Sample question: If you know $P(A|B)$, do you have enough information to compute $P(B|A)$?
Maximum Likelihood Estimation

• **Data:** Observed set $D$ of $\alpha_H$ Heads and $\alpha_T$ Tails

• **Hypothesis:** Binomial distribution

• **Learning:** finding $\theta$ is an optimization problem
  
  – What’s the objective function?

  $$P(D \mid \theta) = \theta^{\alpha_H} (1 - \theta)^{\alpha_T}$$

• **MLE:** Choose $\theta$ to maximize probability of $D$

  $$\hat{\theta} = \arg \max \theta \ P(D \mid \theta)$$

  $$= \arg \max \theta \ \ln P(D \mid \theta)$$
How to get a closed form solution?

\[ \hat{\theta} = \arg\max_\theta \ln P(\mathcal{D} \mid \theta) \]
\[ = \arg\max_\theta \ln \theta^{\alpha_H} (1 - \theta)^{\alpha_T} \]

• Set derivative to zero, and solve!

\[
\frac{d}{d\theta} \ln P(\mathcal{D} \mid \theta) = \frac{d}{d\theta} \left[ \ln \theta^{\alpha_H} (1 - \theta)^{\alpha_T} \right]
\]
\[ = \frac{d}{d\theta} \left[ \alpha_H \ln \theta + \alpha_T \ln(1 - \theta) \right] \]
\[ = \alpha_H \frac{d}{d\theta} \ln \theta + \alpha_T \frac{d}{d\theta} \ln(1 - \theta) \]
\[ = \frac{\alpha_H}{\theta} - \frac{\alpha_T}{1 - \theta} = 0 \]

\[ \hat{\theta}_{MLE} = \frac{\alpha_H}{\alpha_H + \alpha_T} \]
What if I have prior beliefs?

- Billionaire says: Wait, I know that the thumbtack is “close” to 50-50. What can you do for me now?
- You say: I can learn it the Bayesian way...
- Rather than estimating a single $\theta$, we obtain a distribution over possible values of $\theta$

In the beginning

![Graph showing Beta pdf in the beginning](image)

After observations

![Graph showing Beta pdf after observations](image)

Observe flips e.g.: {tails, tails}
Bayesian Learning

Use Bayes rule:

\[ P(\theta \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \theta)P(\theta)}{P(\mathcal{D})} \]

Data Likelihood

Normalization

Prior

Posterior

Or equivalently:

\[ P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta)P(\theta) \]

Also, for uniform priors:

\[ P(\theta) \propto 1 \]

\[ P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta) \]

\[ \rightarrow \text{reduces to MLE objective} \]
MAP: Maximum a Posteriori Approximation

\[ P(\theta \mid \mathcal{D}) \sim \text{Beta}(\beta_H + \alpha_H, \beta_T + \alpha_T) \]

\[ E[f(\theta)] = \int_0^1 f(\theta) P(\theta \mid \mathcal{D}) \, d\theta \]

- As more data is observed, Beta is more certain
- \textbf{MAP}: use most likely parameter to approximate the expectation

\[ \hat{\theta} = \arg \max_{\theta} P(\theta \mid \mathcal{D}) \]

\[ E[f(\theta)] \approx f(\hat{\theta}) \]
What you should know?

• MLE vs MAP and the relationship between the two

• MLE learning and Bayesian learning
  – Thumbtack example
  – Gaussians
The Naïve Bayes Classifier

• Given:
  – Prior $P(Y)$
  – $n$ conditionally independent features $X$ given the class $Y$
  – For each $X_i$, we have likelihood $P(X_i | Y)$

• Decision rule:

$$y^* = h_{NB}(x) = \arg \max_y P(y)P(x_1, \ldots, x_n | y)$$

$$= \arg \max_y P(y) \prod_i P(x_i | y)$$
Subtleties of Naïve Bayes

• What is the hypothesis space?
• What kind of functions can it learn?
• When does it work and when it does not?
  – Correlated features
• MLE vs Bayesian learning of Naïve Bayes
• Gaussian Naïve Bayes
Generative vs. Discriminative Classifiers

• Want to Learn: \( h: X \rightarrow Y \)
  – \( X \) – features
  – \( Y \) – target classes

• Generative classifier, e.g., Naïve Bayes:
  – Assume some **functional form for** \( P(X|Y) \), \( P(Y) \)
  – Estimate parameters of \( P(X|Y) \), \( P(Y) \) directly from training data
  – Use Bayes rule to calculate \( P(Y|X=x) \)
  – This is a ‘**generative**’ model
    - Indirect computation of \( P(Y|X) \) through Bayes rule
    - As a result, can also generate a sample of the data, \( P(X) = \sum_y P(y) P(X|y) \)

• Discriminative classifiers, e.g., Logistic Regression:
  – Assume some **functional form for** \( P(Y|X) \)
  – Estimate parameters of \( P(Y|X) \) directly from training data
  – This is the ‘**discriminative**’ model
    - Directly learn \( P(Y|X) \)
    - But cannot obtain a sample of the data, because \( P(X) \) is not available

\[ P(Y | X) \propto P(X | Y) P(Y) \]
Linear Regression

$h_w(x) = w_1 x + w_0$

$w_1 = \frac{N \sum(x_j y_j) - (\sum x_j)(\sum y_j)}{N \sum(x_j^2) - (\sum x_j)^2}$

$w_0 = \frac{(\sum y_j) - w_1(\sum x_j)}{N}$
Learn $P(Y|X)$ directly!

- Assume a particular functional form

👋 *Not differentiable...*
Logistic Regression

Learn $P(Y|X)$ directly!

- Assume a particular functional form
- Logistic Function
- Aka Sigmoid

$$
\frac{1}{1 + \exp(-z)}
$$
Issues in Linear and Logistic Regression

• Overfitting avoidance: Regularization
  – L1 vs L2 regularization

\[
\begin{align*}
  w_i^{(t+1)} & \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid x^j, w)] \\
  w_i^{(t+1)} & \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid x^j, w)] \right\}
\end{align*}
\]
What you should know about Logistic Regression (LR)

- Gaussian Naïve Bayes with class-independent variances representationally equivalent to LR
  - Solution differs because of objective (loss) function
- In general, NB and LR make different assumptions
  - NB: Features independent given class ! assumption on $P(X|Y)$
  - LR: Functional form of $P(Y|X)$, no assumption on $P(X|Y)$
- LR is a linear classifier
  - decision rule is a hyperplane
- LR optimized by conditional likelihood
  - no closed-form solution
  - concave ! global optimum with gradient ascent
  - Maximum conditional a posteriori corresponds to regularization
- Convergence rates
  - GNB (usually) needs less data
  - LR (usually) gets to better solutions in the limit
Perceptron

\[ o(x_1, \ldots, x_n) = \begin{cases} 
  1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\
  -1 & \text{otherwise.} 
\end{cases} \]

Sometimes we’ll use simpler vector notation:

\[ o(\vec{x}) = \begin{cases} 
  1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\
  -1 & \text{otherwise.} 
\end{cases} \]
From Logistic Regression to the Perceptron: 2 easy steps!

- **Logistic Regression:** (in vector notation): \( y \) is \{0,1\}
  \[
  w = w + \eta \sum_j \left[ y_j^* - p(y_j^*|x_j, w) \right] f(x_j)
  \]

- **Perceptron:** \( y \) is \{0,1\}, \( y(x;w) \) is prediction given \( w \)
  \[
  w = w + \left[ y^* - y(x;w) \right] f(x)
  \]

**Differences?**
- Drop the \( \Sigma_j \) over training examples: **online vs. batch learning**
- Drop the dist’n: **probabilistic vs. error driven learning**
Properties of Perceptrons

- **Separability**: some parameters get the training set perfectly correct

- **Convergence**: if the training is separable, perceptron will eventually converge (binary case)
Problems with the Perceptron

- Noise: if the data isn’t separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)

- Mediocre generalization: finds a “barely” separating solution

- Overtraining: test/validation accuracy usually rises, then falls
  - Overtraining is a kind of overfitting
Multilayer Networks of Sigmoid Units

\[
\text{out}(x) = g\left( w_0 + \sum_k w_k g\left( w_0^k + \sum_i w_i^k x_i \right) \right)
\]
Backpropagation Algorithm

Initialize all weights to small random numbers
Until convergence, Do
   For each training example, Do
      1. Input it to network and compute network outputs
      2. For each output unit $k$
         \[ \delta_k \leftarrow o_k(1-o_k)(t_k-o_k) \]
      3. For each hidden unit $h$
         \[ \delta_h \leftarrow o_h(1-o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k \]
      4. Update each network weight $w_{i,j}$
         \[ w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j} \]
         where $\Delta w_{i,j} = \eta \delta_j x_{i,j}$
Neural networks: What you should know?

- How does it learn non-linear functions?
- Can it learn, for example an XOR function?
  - Draw a neural network for it with appropriate weights
- Backprop
- Overfitting
- What kind of functions can it learn?
- Tradeoff
  - number of hidden units
  - number of layers
Linear SVM

- **Aim:** Learn a large margin classifier
- **Mathematical Formulation:**

  \[
  \text{maximize} \quad \frac{2}{\|w\|}
  \]

  such that

  For \( y_i = +1 \), \( w^T x_i + b \geq 1 \)

  For \( y_i = -1 \), \( w^T x_i + b \leq -1 \)

**Common theme in machine learning:**
LEARNING IS OPTIMIZATION
Solving the Optimization Problem

minimize $L_p(w, b, \alpha_i) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^{n} \alpha_i \left( y_i (w^T x_i + b) - 1 \right)$

s.t. $\alpha_i \geq 0$

Lagrangian Dual Problem

maximize $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$

s.t. $\alpha_i \geq 0$, and $\sum_{i=1}^{n} \alpha_i y_i = 0$
Non-linear SVMs: Feature Space

- General idea: the original input space can be mapped to some higher-dimensional feature space where the training set is separable:

\[ \Phi: x \rightarrow \varphi(x) \]
Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

\[ g(x) = w^T \phi(x) + b = \sum_{i \in SV} \alpha_i \phi(x_i)^T \phi(x) + b \]

- No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.

- A **kernel function** is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

\[ K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j) \]
Nonlinear SVMs: The Kernel Trick

Examples of commonly-used kernel functions:

- **Linear kernel:**
  \[ K(x_i, x_j) = x_i^T x_j \]

- **Polynomial kernel:**
  \[ K(x_i, x_j) = (1 + x_i^T x_j)^p \]

- **Gaussian (Radial-Basis Function (RBF)) kernel:**
  \[ K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \]

- **Sigmoid:**
  \[ K(x_i, x_j) = \tanh(\beta_0 x_i^T x_j + \beta_1) \]

In general, functions that satisfy *Mercer’s condition* can be kernel functions: Kernel matrix should be positive semidefinite.
K-nearest Neighbor

• Distance measure
  – Most common: Euclidean

• Choosing k
  – Increasing k reduces variance, increases bias

• For high-dimensional space, problem that the nearest neighbor may not be very close at all!

• Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.
Nearest Neighbor

• Advantages
  – variable-sized hypothesis space
  – Learning is extremely efficient
    • however growing a good kd-tree can be expensive
  – Very flexible decision boundaries

• Disadvantages
  – distance function must be carefully chosen
  – Irrelevant or correlated features must be eliminated
  – Typically cannot handle more than 30 features
  – Computational costs: Memory and classification-time computation
Locally Weighted Linear Regression: LWLR

• Idea:
  – k-NN forms local approximation for each query point $x_q$
  – Why not form an explicit approximation $\hat{f}$ for region surrounding $x_q$
    • Fit linear function to k nearest neighbors
    • Fit quadratic, ...
    • Thus producing ``piecewise approximation'' to $\hat{f}$
      – Minimize error over k nearest neighbors of $x_q$
      – Minimize error entire set of examples, weighting by distances
      – Combine two above
Bias-Variance-Noise Decomposition

\[ E[(h(x*) - y*)^2] = E[h(x*)^2] - 2h(x*)y* + y*^2 \]

\[ = E[h(x*)^2] - 2E[h(x*)]E[y*] + E[y*^2] \]

\[ = E[(h(x*) - \overline{h(x*)})^2] + \overline{h(x*)}^2 \]

\[ - 2\overline{h(x*)}f(x*) \]

\[ + E[(y* - f(x*))^2] + f(x*)^2 \]

\[ = E[(h(x*) - \overline{h(x*)})^2] + \overline{h(x*)}^2 - f(x*)^2 \]

\[ + E[(y* - f(x*))^2] \]

\[ = \text{Var}(h(x*)) + \text{Bias}(h(x*))^2 + E[\varepsilon^2] \]

\[ = \text{Var}(h(x*)) + \text{Bias}(h(x*))^2 + \sigma^2 \]

Expected prediction error = Variance + Bias^2 + Noise^2
Bias, Variance, and Noise

• Variance: $E[ (h(x^*) - h(x^*))^2 ]$
  Describes how much $h(x^*)$ varies from one training set $S$ to another

• Bias: $[h(x^*) - f(x^*)]$
  Describes the average error of $h(x^*)$.

• Noise: $E[ (y^* - f(x^*))^2 ] = E[\varepsilon^2] = \sigma^2$
  Describes how much $y^*$ varies from $f(x^*)$
Bias/Variance Tradeoff

• \((\text{bias}^2 + \text{variance})\) is what counts for prediction

• Often:
  – low bias \(\Rightarrow\) high variance (too many parameters)
  – low variance \(\Rightarrow\) high bias (too few parameters)

• Tradeoff:
  – \(\text{bias}^2\) vs. variance
Bagging: Bootstrap Aggregation

• Leo Breiman (1994)
• Take repeated bootstrap samples from training set $D$.
• **Bootstrap sampling**: Given set $D$ containing $N$ training examples, create $D'$ by drawing $N$ examples at random with replacement from $D$.

• Bagging:
  – Create $k$ bootstrap samples $D_1 \ldots D_k$.
  – Train distinct classifier on each $D_i$.
  – Classify new instance by majority vote / average.
AdaBoost

1. Initialize the data weighting coefficients \( \{w_n\} \) by setting \( w_n^{(1)} = 1/N \) for \( n = 1, \ldots, N \).

2. For \( m = 1, \ldots, M \):
   
   (a) Fit a classifier \( y_m(x) \) to the training data by minimizing the weighted error function
   
   \[
   J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)
   \]  
   (14.15)

   where \( I(y_m(x_n) \neq t_n) \) is the indicator function and equals 1 when \( y_m(x_n) \neq t_n \) and 0 otherwise.

   (b) Evaluate the quantities
   
   \[
   \epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}
   \]  
   (14.16)

   and then use these to evaluate
   
   \[
   \alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}
   \]  
   (14.17)

(c) Update the data weighting coefficients

\[
\begin{align*}
  w_n^{(m+1)} &= w_n^{(m)} \exp \left\{ \alpha_m I(y_m(x_n) \neq t_n) \right\} 
\end{align*}
\]  
(14.18)