Goals For These Slides

- A quick overview of key techniques for learning linear classifiers
- Why study linear classifiers in the neural network future?
  - Key concepts are (re)used in fancier classifiers
  - Fast and reliable to train (no local optima)
  - (more) interpretable output
  - Still good enough for many purposes
Classification

• Supervised learning framework
• Features can be anything
• Targets are discrete classes:
  – Safe mushrooms vs. poisonous
  – Malignant vs. benign
  – Good credit risk vs. bad
• Can we treat classes as numbers?
  – Single class?
  – Multi class?

Representing Classes

• Interpret $t^{(i)}$ as the probability that the $i^{th}$ element is in a particular class
• Classes usually disjoint
• For multiclass, $t^{(i)}$ may be a vector
• $t^{(i)[j]}=t^{(i)}_j=1$ if $i^{th}$ element is in class $j$, 0 OTW

• Notation: For convenience, we will sometimes refer to the “raw” variables $x$, rather than the features as seen through the lens of our features, $\phi$
What is a Linear Discriminant?

• Simplest kind of classifier, a linear threshold unit (LTU):

\[
y(x) = \begin{cases} 
1 & \text{if } w_1 x_1 + \cdots + w_n x_n \geq w_0 \\
0 & \text{otherwise}
\end{cases}
\]

• Note: sometimes use +1/-1 instead of 1/0 for mathematical convenience
• A linear discriminant is an n-1 dimensional hyperplane
• \(w\) is orthogonal to this
• Four algorithms for linear decision boundaries:
  – Directly learn the LTU using, e.g., a perceptron
  – Learn the conditional distribution: Logistic regression
  – Learn a distribution:
    – Naïve Bayes
    – Linear discriminant analysis (LDA)

Decision Boundaries

• A classifier can be viewed as partitioning the input space or feature space \(X\) into decision regions

• A linear threshold unit always produces a linear decision boundary. A set of points that can be separated by a linear decision boundary is linearly separable.
What can be expressed?

- Examples of things that can be expressed (Assume n Boolean (0/1 features))
  - Conjunctions:
    - $x_1 \land x_3 \land x_4$: $1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 3$
    - $x_1 \land \neg x_3 \land x_4$: $1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 2$
  - at-least-m-of-n
    - at-least-2-of($x_1, x_2, x_4$)
      - $1 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 + 1 \cdot x_4 \geq 2$
  - Examples of things that cannot be expressed:
    - Non-trivial disjunctions:
      - $(x_1 \lor x_3) + (x_3 \lor x_4)$
    - Exclusive-Or
      - $(x_1 \land \neg x_3) + (\neg x_1 \land x_3)$

Non-linearly separable example (XOR)
**Multiclass**

- Typically study binary classification, with the assumption that multiclass is a simple generalization
- How do we do that?

- \(O(k^2)\) one vs. one classifiers
- \(k\) one vs. rest classifiers

- Jointly learn \(k\) outputs, interpret outputs (after some processing) as probabilities of class membership (see digit classification on HW coding assignment)

---

**Perceptron**

\[ y = \text{sgn}(\mathbf{w}^T \mathbf{x}) \]
How a Perceptron Classifies

\[ y = f(x, w) = h\left(\sum_{i=1}^{n} w_i x_i\right) \]

Perceptron Learning

- We are given a set of inputs \( x^{(1)} \ldots x^{(n)} \)
- \( t^{(1)} \ldots t^{(n)} \) is a set of target outputs (Boolean) \{-1, 1\}
- \( w \) is our set of weights
- output of perceptron = \( \text{sgn}(w^T x) \)
- \( \text{Perceptron\_error}(x^{(i)}, w) = -\text{sgn}(w^T x) * t^{(i)} \)
  - +1 when perceptron is incorrect
  - -1 when perceptron is correct
- Goal: Pick \( w \) to optimize:
  \[ \min_w \sum_{i \in \text{misclassified}} \text{perceptron\_error}(x^{(i)}, w) \]
Update Rule

Repeat until convergence:

\[
\forall i \in \text{misclassified} \forall j : w_j \leftarrow w_j + \alpha x_j^{(i)} t^{(i)}
\]

“Learning Rate”
(can be any constant)

• i iterates over samples
• j iterates over weights
• Unlike neural network gradient descent
  perceptron updates weights asynchronously
  (synchronous also works)

Perceptron Learning Properties

(LTU Properties)

• Good news:
  – If there exists a set of weights that will correctly classify
    every example, the perceptron learning rule will find it
  – Does not depend on step size!
• Bad news:
  – Perceptrons can represent only a small class of functions,
    “linearly separable,” functions (LTU property)
  – May oscillate if not separable
  – No obvious generalization for multiclass
Logistic Regression

- In logistic regression, we learn the conditional distribution $P(t|x)$.
- Let $p_t(x; w)$ be our estimate of $P(t|x)$, where $w$ is a vector of adjustable parameters.
- Assume there are two classes, $t = 0$ and $t = 1$ and
  \[ p_1(x; w) = \frac{e^{w^Tx}}{1 + e^{w^Tx}} = \frac{1}{1 + e^{-w^Tx}} \]
  \[ p_0(x; w) = 1 - p_1(x; w) = \frac{1}{1 + e^{w^Tx}} \]
- This is equivalent to
  \[ \log \frac{p_1(x; w)}{p_0(x; w)} = w^Tx \]
- IOW, the log odds of class 1 is a linear function of $x$

Why this form?

- One reason: transforms a linear function in the range (-∞, +∞) to [0,1]
Constructing a Learning Algorithm

- Find the probability distribution $h$ that is most likely, given the data.

$$\arg\max_{h} P(h|X) = \arg\max_{h} \frac{P(X|h)P(h)}{P(X)} \text{ by Bayes' Rule}$$

$$= \arg\max_{h} P(X|h)P(h) \text{ because } P(X) \text{ doesn't depend on } h$$

$$= \arg\max_{h} P(X|h) \text{ if we assume } P(h) \text{ is uniform}$$

$$= \arg\max_{h} \log P(X|h) \text{ because } \log \text{ is monotone }$$

- The likelihood function views $P(X|h)$ as a function of the parameters in the model. In this case, our parameters are the weights, $w$.
- The log likelihood is a commonly used objective function for learning algorithms. It is denoted $L(w;X)$.
- The $w$ that maximizes the likelihood of the training data is called the maximum likelihood estimator.

Computing the Likelihood

(Full derivation not presented, but included for reference)

- Assume that each training example $(x^i, t^i)$ is drawn from the same underlying (but unknown) probability distribution $P(x,t)$. This means that the log likelihood of $X$ is the sum of the log likelihoods of the individual training examples:

$$\log P(X|h) = \log \prod_{i} P(x_i, t_i|h)$$

$$\log P(X|h) = \sum_{i} \log P(x_i, t_i|h)$$

- Any joint distribution $P(a,b)$ can be factored as $P(a|b)P(b)$. So, we have:

$$\arg\max_{h} \log P(X|h) = \arg\max_{h} \sum_{i} \log P(x_i, t_i|h)$$

$$= \arg\max_{h} \sum_{i} \log P(t_i|x_i,h)P(x_i|h)$$

- Now, $P(x|h)=P(x)$, because it does not depend on $h$.

$$\arg\max_{h} \log P(X|h) = \arg\max_{h} \sum_{i} \log P(t_i|x_i,h)P(x_i)$$

$$= \arg\max_{h} \sum_{i} \log P(t_i|x_i,h)$$

- Hence, the log likelihood of $X$ is the sum of the log conditional likelihood of the individual data points.
Log Likelihood for Conditional Probability Estimators

- We can express the log likelihood in a compact form.
- Take an example ($x^0, t^0$)
  - If $y^0 = 0$, the log likelihood is $\log[1 - p_i(x; w)]$
  - If $y^0 = 1$, the log likelihood is $\log p_i(x; w)$
- These two are mutually exclusive, so we can combine them to get:

$$L(w; x^0, t) = \log p(t^0 \mid x^0, w) = (1 - t^0)\log[1 - p_i(x^0; w)] + t^0 \log p_i(x^0; w)$$

- The goal of our learning algorithm will be to find $w$ to maximize:

$$L(w; X, t)$$

Computing the Gradient

$$\frac{\partial L(w)}{\partial w} = \sum \frac{\partial}{\partial w}L(w; t^0, x^0)$$

$$\frac{\partial}{\partial w}L(w; t^0, x^0) = \frac{\partial}{\partial w}(1 - t^0)\log[1 - p_i(x^0; w)] + t^0 \log p_i(x^0; w))$$

$$= (1 - t^0) \frac{1}{1 - p_i(x^0; w)} \left( \frac{\partial p_i(x^0; w)}{\partial w} \right) + t^0 \frac{1}{p_i(x^0; w)} \left( \frac{\partial p_i(x^0; w)}{\partial w} \right)$$

$$= \left[ \frac{1}{p_i(x^0; w)} - \frac{1}{1 - p_i(x^0; w)} \right] \left( \frac{\partial p_i(x^0; w)}{\partial w} \right)$$

$$= \left[ \frac{1 - p_i(x^0; w)}{p_i(x^0; w)} + \frac{p_i(x^0; w)}{1 - p_i(x^0; w)} \right] \left( \frac{\partial p_i(x^0; w)}{\partial w} \right)$$

$$= \left[ \frac{1 - p_i(x^0; w)}{p_i(x^0; w)} \frac{\partial p_i(x^0; w)}{\partial w} \right]$$

$$= \left[ \frac{1 - p_i(x^0; w)}{p_i(x^0; w)} \frac{\partial p_i(x^0; w)}{\partial w} \right]$$
Gradient cont.

- Recall the form of $p_1$:
  \[ p_1(x^{(i)}; w) = \frac{e^{w'x^{(i)}}}{1 + e^{w'x^{(i)}}} = \frac{1}{1 + e^{-w'x^{(i)}}} \]
- So we get:
  \[
  \frac{\partial p_1(x^{(i)}; w)}{\partial w_j} = \frac{-1}{(1 + e^{-w'x^{(i)}})^2} \frac{\partial}{\partial w_j} (1 + e^{-w'x^{(i)}})
  = \frac{1}{(1 + e^{-w'x^{(i)}})^2} e^{w'x^{(i)}} \frac{\partial}{\partial w_j} (w'x^{(i)})
  = \frac{1}{(1 + e^{-w'x^{(i)}})^2} e^{w'x^{(i)}} (x^{(i)_j})
  = p_1(x^{(i)}; w)(1 - p_1(x^{(i)}; w)) x^{(i)}_j
  \]
  Recall: \[ p_1(x; w) = 1 - p_1(x; w) - \frac{e^{-w'x}}{1 + e^{-w'x}} \]

Gradient cont.

- The gradient of the log likelihood for a single point is thus:
  \[
  \frac{\partial}{\partial w_j} L(w; x^{(i)}, t^{(i)}) = \left[ \frac{t^{(i)} - p_1(x^{(i)}; w)}{p_1(x^{(i)}; w)(1 - p_1(x^{(i)}; w))} \right] \frac{\partial p_1(x^{(i)}; w)}{\partial w_j}
  = \left[ \frac{t^{(i)} - p_1(x^{(i)}; w)}{p_1(x^{(i)}; w)(1 - p_1(x^{(i)}; w))} \right] p_1(x^{(i)}; w)(1 - p_1(x^{(i)}; w)) x^{(i)}_j
  = (t^{(i)} - p_1(x^{(i)}; w)) x^{(i)}_j
  \]
- The overall gradient is:
  \[
  \frac{\partial L(w)}{\partial w_j} = \sum_i (t^{(i)} - p_1(x^{(i)}; w)) x^{(i)}_j
  \]
  Compare w/ perceptron rule!
Batch Ascent/Descent

- Logistic regression w/training set \( \{(x^{(i)}, t^{(i)})\}, i = 1..N \)
  Repeat until convergence { for every j
  \[
  w^{(t+1)}_j = w^{(t)}_j + \alpha \sum_{i=1}^{N} (t^{(i)} - p(x^{(i)}, w^{(t)})) x^{(i)}_j
  \]
  t++}

- Perceptron (synchronous version):
  Repeat until convergence { for every j
  \[
  w^{(t+1)}_j = w^{(t)}_j + \alpha \sum_{i \in \text{misclassified}} x^{(i)}_j
  \]
  t++}

NB: \((t)\) is a time/iteration index, which indicates that updates are done synchronously, i.e., all weights on the RHS are frozen until all updates are computed, then all weights are simultaneously updated together.

Comments on (A)synchronous Updates

- Perceptron training rule presented as an “online” (asynchronous) algorithm (one update per datum)
  - Q: Can you do perceptron learning in batch?
  - A: Yes, but you don’t need to.

- Logistic regression presented as a batch (synchronous) algorithm
  - Q: Can you do logistic regression “online”?
  - A: Yes, if you are careful (small step size)
Logistic Regression for K > 2

(Not Presented, but for reference)

• To handle K > 2 classes, we make one class the 'reference' class. Suppose it is class K. Then we represent each of the other classes as a logistic function of the odds of class k versus class K:

\[
\begin{align*}
\frac{P(y = 1|x)}{P(y = K|x)} &= e^{\theta_1 \cdot x} \\
\frac{P(y = 2|x)}{P(y = K|x)} &= e^{\theta_2 \cdot x} \\
&\vdots \\
\frac{P(y = k-1|x)}{P(y = K|x)} &= e^{\theta_{k-1} \cdot x} \\
\frac{P(y = K|x)}{P(y = K|x)} &= e^{\theta_{K-1} \cdot x}
\end{align*}
\]

• The conditional probability for class \( k \neq K \) is

\[
P(y = k|x) = \frac{e^{\theta_k \cdot x}}{1 + \sum_{j \neq k} e^{\theta_j \cdot x}}
\]

• and for class \( k = K \):

\[
P(y = K|x) = \frac{1}{1 + \sum_{j \neq K} e^{\theta_j \cdot x}}
\]

Summary of Logistic Regression

• Learns the **Conditional Probability Distribution** \( P(y|x) \)

• Very simple expression for gradient permits local search
  – Begin with initial weight vector.
  – Gradient ascent to maximize objective function – finds global optimum
  – May be done online or in batch

• Advantages over perceptron:
  – Does not assume separable data
  – Gives probabilities

• Disadvantages: May be slower to train
What We Already Know

- Linear Threshold Unit (LTU)
  - Tries to discover a linear function (in feature space) that separates positive and negative examples
  - Example: Perceptron

- Logistic Regression
  - Maximizes log likelihood
  \[
  \log \frac{\rho_1(x; w)}{\rho_0(x; w)} = w^T x
  \]

Density Estimation

- Instead of directly learning classifier, model the full probability distribution of both features and labels
- Potentially more challenging learning problem, but also more versatile
Discrete Case

- Suppose we know $P(X_1...X_n,T)$
- Assuming:
  - Binary loss function
  - Choices: $t_0, t_1$
- Favor $t_0$ when $P(t_0|x_1...x_n) > P(t_1|x_1...x_n)$
- Use definition of conditional probability:
  $$P(t_0 | x_1...x_n) = \frac{P(t_0 x_1...x_n)}{P(x_1...x_n)}$$

So, are we done???

- How many parameters needed for joint?
- Is this practical?
- Simplification (Naïve Bayes):
  $$P(X_1...X_n | t) = \prod_i P(X_i | t)$$
- More general approach: Use a Bayes net
Naïve Bayes is a linear method!

- Choose class 1 when:
  
  \[
  \prod_{i=1}^{n} P(x_i | t_1) P(t_1) > \prod_{i=1}^{n} P(x_i | t_0) P(t_0)
  \]

- Fundamentally same expressive power as other linear methods

What about continuous features?

- Could be arbitrarily complicated to model joint distribution over label and features without additional assumptions

- As usual, we find that making a Normal (Gaussian) density assumption leads to a tractable solution

- Nota bene: This part may be a bit opaque to you you are rusty on your stats. Don’t sweat the details if some of this feels uncomfortable
Linear (Gaussian) Discriminant Analysis

- In LDA, we learn the distribution $P(x|t)$
- We assume that $x$ is continuous
- We assume $P(x|t)$ is distributed according to a multivariate normal distribution and $P(t)$ is a discrete distribution
- Nota bene: LDA can also mean “Latent Dirichlet Allocation”, which is something different

Estimating the MVG parameters

- Given a set of data points $\{x^{(i)}\}$, the maximum likelihood estimates for the parameters (means and covariance) of the MVG are:

\[
\hat{\mu} = \frac{1}{N} \sum_{i} x^{(i)}
\]

\[
\hat{\Sigma} = \frac{1}{N-1} \sum_{i} (x^{(i)} - \hat{\mu})(x^{(i)} - \hat{\mu})^T
\]
Putting it all together in LDA

- Also called Gaussian Discriminant Analysis
- Here
  - $t \sim \text{Bernoulli}(w)$ unconditional (prior) label probability
  - $x|t=0 \sim N(\mu_0, \Sigma)$
  - $x|t=1 \sim N(\mu_1, \Sigma)$  

- Writing this out, we get:

$$
\begin{align*}
  p(x|t=0) &= \frac{1}{(2\pi)^{n/2} |\Sigma|^1/2} \exp \left[ -\frac{1}{2} (x - \mu_0)^T \Sigma^{-1} (x - \mu_0) \right] \\
  p(x|t=1) &= \frac{1}{(2\pi)^{n/2} |\Sigma|^1/2} \exp \left[ -\frac{1}{2} (x - \mu_1)^T \Sigma^{-1} (x - \mu_1) \right]
\end{align*}
$$

Called the Class Conditional densities

Picking A Class

- We again use Bayes rule:

$$
P(t|X) = \frac{P(X|t)P(t)}{P(X)}
$$

Prior class probability

Prior feature probability (ignored)
The Beauty of Homoscedasticity

- Recall we assumed $\Sigma$ same for all classes
- When is $P(t_0|x)>P(t_1|x)$?

$$
\frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{\sqrt{\det{\Sigma}}} \exp\left(-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0)\right) \rho(t_0) >
\frac{1}{(2\pi)^{\frac{n}{2}}} \frac{1}{\sqrt{\det{\Sigma}}} \exp\left(-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)\right) \rho(t_1)
$$

$$
-(x-\mu_0)^T \Sigma^{-1}(x-\mu_0) + k_o > -(x-\mu_1)^T \Sigma^{-1}(x-\mu_1) + k_o
$$

Linear!!!

(Distribute and observe that quadratic terms cancel out)

Example

The decision boundary is at $p(y=1|x) = 0.5$
**Homoscedastic LDA Discussion**

- For multiclass, gives convex decision boundaries
- Nice because it makes classification easy
  (easy to use geometric data structures such as Voronoi diagrams)

- How realistic is this?
- What do we give up?

**Heteroscedastic Distributions**

(assuming uniform class priors, in this example)
Comparing LTU, LR, LDA

- People of wonder about relative merits of
  - direct classifiers (like LTU) versus
  - conditional models (like LR) versus
  - generative models (like LDA, NB)

LDA vs LR

- What is the relationship?
  - In LDA, it turns out the $p(t|x)$ can be expressed as a logistic function where the weights are some function of $\mu_1, \mu_2,$ and $\Sigma$, i.e., can convert LDA solutions to LR solutions.
  - But, the converse is NOT true. If $p(t|x)$ is a logistic function, that does not imply $p(x|t)$ is MVG.
- LDA makes stronger modeling assumptions than LR
  - when these modeling assumptions are correct, LDA will perform better
    - LDA is asymptotically efficient: in the limit of very large training sets, there is no algorithm that is strictly better than LDA.
  - however, when these assumptions are incorrect, LR is more robust
    - weaker assumptions, more robust to deviations from modeling assumptions
    - if the data are non-Gaussian, then in the limit, LR outperforms LDA.
    - For this reason, LR is a more commonly used algorithm.
Issues

• **Statistical efficiency**: if the generative model is correct, then it usually gives better accuracy, especially for small training sets.
• **Computational efficiency**: generative models typically are the easiest to compute. In LDA, we estimated the parameters directly, no need for gradient ascent.
• **Robustness to changing loss function**: Both generative and conditional models allow the loss function to change without re-estimating the model. This is not true for direct LTU methods.
• **Robustness to model assumptions**: A generative model usually performs poorly when the assumptions are violated.
• **Robustness to missing values and noise**: In many applications, some of the features $x_i^j$ may be missing or corrupted for some training examples. Generative models provide better ways of handling this than non-generative models.

Conclusions

• Four linear methods
  – Perceptron
  – Logistic regression
  – Naïve Bayes
  – Linear Discriminant Analysis

• Perceptrons are simple, fast
• LR, NB give probabilities
• LDA, NB model the data
  – Handle missing features
  – Best results when modeling assumptions are accurate