Linear Classification

Ron Parr
CPS 271

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, $\mathbf{t}^{(i)}$ is a vector
- $\mathbf{t}^{(i)}[j]=\mathbf{t}^{(i)}_j=1$ 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, $\boldsymbol{\phi}$

What is a Linear Disciminant?

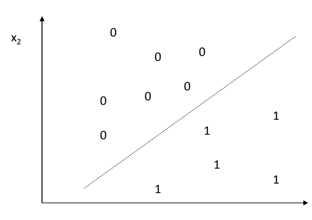
- Simplest kind of classifer, a **linear threshold unit (LTU):**

$$y(\mathbf{x}) = \begin{cases} 1 & if \;\; w_1 x_1 + \cdots + \theta_n w_n \geq w_0 \\ 0 & otherwise \end{cases}$$

- We sometimes assume $w_0=1$, so $y(\mathbf{x})=\mathbf{w}^T\mathbf{x}$
- A linear discriminant is an n-1 dimensional hyperplane
- **w** is orthogonal to this
- We'll look at three algorithms, all of which learn linear decision boundaries:
  - Directly learn the LTU: Using Least Mean Square (LMS) algorithm
  - Learn the conditional distribution: Logistic regression
  - Learn the joint distribution: Linear discriminant analysis (LDA)

Decision Boundaries

- A classifier can be viewed as partitioning the input space or feature space X into decision regions

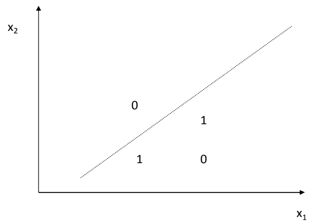


- A linear threshold unit always produces a linear decision boundary. A set of points that can be separated by a linear decision boundary is **linearly separable**.

What can be expressed?

- Examples of things that can be expressed
  (Assume n boolean (0/1 features)
  - Conjunctions:
    - $x_1 \wedge x_3 \wedge x_4$: $1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 3$
    - $x_1 \wedge \neg x_3 \wedge x_4$: $1 \cdot x_1 + 0 \cdot x_2 + -1 \cdot x_3 + 1 \cdot x_4 \geq 2$
  - 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:
    - $(x_1 \wedge x_3) + (x_3 \wedge x_4)$
  - Exclusive-Or
    - $(x_1 \wedge \neg x_2) + (\neg x_1 \wedge x_2)$

Non-linearly separable example

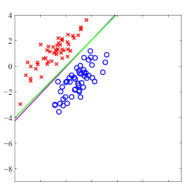


Multiclass

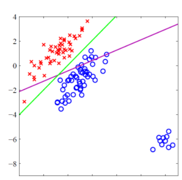
- k classes
- $O(k^2)$ one vs. one classifiers
  - Expensive
  - May not be consistent
- k-1 one vs. rest classifiers
  - Less expensive
  - Still may not be consistent
- K linear functions
  - Assign x to class j if $\mathbf{w}_j^T\mathbf{x} > \mathbf{w}_i^T\mathbf{x}$ for all i
  - Gives convex, singly connected decision regions
  - How to pick the linear functions?

Why not use regression?

- Regression minimizes sum of squared errors on target function
- Gives strong influence to outliers



Note: Class labels are in Z dimension



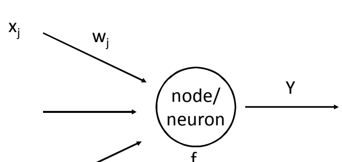
Magenta = linear regression

The "Neural" Story (Part I)

- Nice to justify machine learning w/nature
- Naïve introspection works badly
- Neural model biologically plausible

- Single neuron, linear threshold unit = perceptron
- (Longer rant on this later…)

Perceptron



f is a simple step function (sgn)

Perceptron Learning

- We are given a set of inputs $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(n)}$
- $t^{(1)} \ldots t^{(n)}$ is a set of target outputs (boolean) {-1,1}
- $\mathbf{w}$ is our set of weights
- output of perceptron = $\mathbf{w}^T\mathbf{x}$
- Perceptron_error($\mathbf{x}^{(i)}$, w) = -net($\mathbf{x}^{(i)}$,$\mathbf{w}$) $t^{(i)}$
- Goal: Pick w to optimize:

$$\min_{\mathbf{w}} \sum_{i \in \text{misclassified}} perceptron\_error(\mathbf{x}^{(i)}, \mathbf{w})$$

Update Rule

Repeat until convergence:

$$\underset{i \in \text{misclassified}}{\forall} \; \underset{j}{\forall} : w_j \leftarrow w_j + \alpha x_j^{(i)} t^{(i)}$$

"Learning Rate"
(can be any constant)

- i iterates over samples
- j iterates over weights

http://neuron.eng.wayne.edu/java/Perceptron/New38.html

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
  - May oscillate if not separable
  - No obvious generalization for multiclass

Logistic Regression

- In logistic regression, we learn the conditional distribution P(t|**x**)
- Let $p_t(\mathbf{x}; \mathbf{W})$ 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

$$p_1(\mathbf{x};\mathbf{w}) = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}}$$
$$p_0(\mathbf{x};\mathbf{w}) = 1 - p_1(\mathbf{x};\mathbf{w})$$

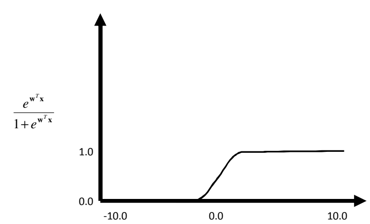
- This is equivalent to

$$\log \frac{p_1(\mathbf{x};\mathbf{w})}{p_0(\mathbf{x};\mathbf{w})} = \mathbf{w}^T\mathbf{x}$$

- 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 be positive and sum to 1 so that it can represent a probability



Constructing a Learning Algorithm

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

$$\begin{aligned}
\arg\max_{h_w} P(h_w \mid X) &= \arg\max_{h_w} \frac{P(X \mid h_w)P(h_w)}{P(X)} && \text{by Bayes' Rule} \\
&= \arg\max_{h_w} P(X \mid h_w)P(h_w) && \text{because P(X) doesn't depend on h} \\
&= \arg\max_{h_w} P(X \mid h_w) && \text{if we assume P(h) is uniform} \\
&= \arg\max_{h_w} \log P(X \mid h_w) && \text{because log is monotone}
\end{aligned}$$

- The **likelihood function** views $P(X|h_w)$ as a function of the parameters in the model. In this case, our parameters are the weights, **w.**
  $L(w;X) = P(X|h_w)$
- 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**

Log Likelihood for Conditional Probability Estimators

- We can express the log likelihood in a compact from called the **cross-entropy**
- Take an example $(\mathbf{x}^{(i)}, t^{(i)})$
  - if $y^{(i)} = 0$, the log likelihood is $\log(1-p_1(\mathbf{x}; w))$
  - if $y^{(i)} = 1$, the log likelihood is $\log p_1(\mathbf{x}; \theta)$
- These two are mutually exclusive, so we can combine them to get:

$$\ell(\mathbf{w};\mathbf{x}^{(i)},t) = \log P(t^{(i)} \mid \mathbf{x}^{(i)},\mathbf{w}) = (1-t^{(i)})\log\left[1-p_1(\mathbf{x}^{(i)};\mathbf{w})\right] + t^{(i)}\log p_1(\mathbf{x}^{(i)};\mathbf{w})$$

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

$$J(\mathbf{w}) = \ell(\mathbf{w};\mathbf{X},\mathbf{t})$$

Computing the Gradient

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_j} = \sum_i \frac{\partial}{\partial \mathbf{w}_j} \ell(\theta; t^{(i)}, \mathbf{x}^{(i)})$$

$$\frac{\partial}{\partial \mathbf{w}_j} \ell(\mathbf{w}; t^{(i)}; \mathbf{x}^{(i)}) = \frac{\partial}{\partial \mathbf{w}_j}((1-t^{(i)}) \log[1 - p_1(\mathbf{x}^{(i)}; \mathbf{w})] + t^{(i)} \log p_1(\mathbf{x}^{(i)}; \mathbf{w}))$$

$$= (1-t^{i}) \frac{1}{1-p_1(\mathbf{x}^{(i)};\mathbf{w})} \left(-\frac{\partial p_1(\mathbf{x}^{(i)}; \mathbf{w})}{\partial \mathbf{w}_j} \right) + t^{(i)} \frac{1}{p_1(\mathbf{x}^{(i)};\mathbf{w})} \left(\frac{\partial p_1(\mathbf{x}^{(i)}; \mathbf{w})}{\partial \mathbf{w}_j} \right)$$

$$= \left[\frac{t^{(i)}}{p_1(\mathbf{x}^{(i)};\mathbf{w})} - \frac{(1-t^{(i)})}{1-p_1(\mathbf{x}^{(i)};\mathbf{w})} \right] \left(\frac{\partial p_1(\mathbf{x}^{(i)}; \mathbf{w})}{\partial \mathbf{w}_j} \right)$$

$$= \left[\frac{t^{(i)}(1-p_1(\mathbf{x}^{(i)};\mathbf{w})) - (1-t^{(i)})\, p_1(\mathbf{x}^{(i)};\mathbf{w})}{p_1(\mathbf{x}^{(i)};\mathbf{w})(1-p_1(\mathbf{x}^{(i)};\mathbf{w}))} \right] \left(\frac{\partial p_1(\mathbf{x}^{(i)}; \mathbf{w})}{\partial \mathbf{w}_j} \right)$$

$$= \left[\frac{t^{(i)} - p_1(\mathbf{x}^{(i)};\mathbf{w})}{p_1(\mathbf{x}^{(i)};\mathbf{w})(1-p_1(\mathbf{x}^{(i)};\mathbf{w}))} \right] \left(\frac{\partial p_1(\mathbf{x}^{(i)}; \mathbf{w})}{\partial \mathbf{w}_j} \right)$$

Gradient cont.

- Another way of writing the logistic regression function is:

$$p_1(\mathbf{x}^{(i)}; \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}^{(i)}}}$$

- So we get:

$$\frac{\partial p_1(\mathbf{x}^{(i)}; \mathbf{w})}{\partial \mathbf{w}_j} = \frac{1}{(1+e^{-\mathbf{w}^T \mathbf{x}^{(i)}})^2} \frac{\partial}{\partial \mathbf{w}_j} (1 + e^{-\mathbf{w}^T \mathbf{x}^{(i)}})$$

$$= \frac{1}{(1+e^{-\mathbf{w}^T \mathbf{x}^{(i)}})^2} e^{-\mathbf{w}^T \mathbf{x}^{(i)}} \frac{\partial}{\partial \mathbf{w}_j} (-\mathbf{w}^T \mathbf{x}^{(i)})$$

$$= \frac{1}{(1+e^{-\mathbf{w}^T \mathbf{x}^{(i)}})^2} e^{-\mathbf{w}^T \mathbf{x}^{i}} (-x^{(i)}{}_j)$$

$$= p_1(\mathbf{x}^{(i)}; \mathbf{w})(1 - p_1(\mathbf{x}^{(i)}; \mathbf{w}))x^{(i)}{}_j$$

Gradient cont.

- The gradient of the loglikelihood for a single point is:

$$\frac{\partial}{\partial \mathbf{w}_j} \ell(\mathbf{w}; \mathbf{x}^{(i)}, t^{(i)}) = \left[\frac{t^{(i)} - p_1(\mathbf{x}^{(i)};\mathbf{w})}{p_1(\mathbf{x}^{(i)};\mathbf{w})(1-p_1(\mathbf{x}^{(i)};\mathbf{w}))} \right] \left(\frac{\partial p_1(\mathbf{x}^{(i)}; \mathbf{w})}{\partial \mathbf{w}_j} \right)$$

$$= \left[\frac{t^{(i)} - p_1(\mathbf{x}^{(i)};\mathbf{w})}{p_1(\mathbf{x}^{(i)};\mathbf{w})(1-p_1(\mathbf{x}^{(i)};\mathbf{w}))} \right] p_1(\mathbf{x}^{(i)}; \mathbf{w})(1 - p_1(\mathbf{x}^{(i)}; \mathbf{w}))x^{(i)}{}_j$$

$$= (t^{(i)} - p_1(\mathbf{x}^{(i)}; \mathbf{w}))x^{(i)}{}_j$$

- The overall gradient is:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_j} = \sum_i (t^{(i)} - p_1(\mathbf{x}^{i}; \mathbf{w}))x^{(i)}{}_j$$

Compare w/percepton rule!

Summary of Logistic Regression

- Learns the **Conditional Probability Distribution** P(t|**x**)
- No closed form solution
- Very simple expression for gradient
  - Solve by local search:
    - Begins with initial weight vector.
    - Gradient ascent to maximize objective function.
    - Objective function is the **log likelihood** of the data
    - Algorithm seeks the probability distribution P(t|**x**) that is most likely given the data.
- May be done online or in batch
- Can be used with acceleration methods (Newton-Raphson, etc.)

What We Already Know

- Linear Threshold Unit (LTU)
  - Tries to discover a linear function (in feature space) that separates positive and negative examples
- Logistic Regression
  - Uses regression to estimate the function

$$\log \frac{p_1(\mathbf{x}; \mathbf{w})}{p_0(\mathbf{x}; \mathbf{w})} = \mathbf{w}^T \mathbf{x}$$

Density Estimation

- Basic unsupervised learning technique
- Discussed here in context of classification
- Idea: Estimate joint probability of features and class labels

Discrete Case

- Suppose we know $P(X_1 \ldots X_n T)$
- How do we get this?
- Maximum likelihood estimate comes from counting (relative frequency)
- Bernouli distribution

- We see a new $x_1 \ldots x_n$
- What is our guess for t?

Betting on y

- Assuming:
  - Binary loss function
  - Choices: t0, t1
- Favor t0 when $P(t0|x_1 \ldots x_n) > P(t_1|x_1 \ldots x_n)$
- Use definition of conditional probability:

$$P(t0 \mid x_1 \ldots x_n) = \frac{P(t0 x_1 \ldots x_n)}{P(x_1 \ldots x_n)}$$

So, are we done???

- How many parameters needed for joint?
- Is this practical?
- Simplification (Naïve Bayes):

$$P(X_1 \ldots X_n \mid t) = \prod_i P(X_i \mid t)$$

Q: How is this more practical?

Naïve Bayes in Action

- Spam filtering
- X1…Xn: Spam related features
- t: Spam label
- Combine Bayes Rule w/Naïve Bayes:

$$P(t \mid X_1 \ldots X_n) = \frac{P(X_1 \ldots X_n \mid t) P(t)}{P(X_1 \ldots X_n)}$$

$$= \frac{\prod_i P(X_i \mid t) P(t)}{P(X_1 \ldots X_n)}$$

Things to note:

Do we worry about P(X1…Xn)?

Influence of P(t)?

Is Naïve Bayes Reasonable?

- Are features correlated within classes?

- How would it hurt us if they were?

- More on this when we discuss Bayesian networks

Linear 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

Estimating the MVG parameters

- Given a set of data points $\{\mathbf{x}^1,\ldots,\mathbf{x}^N\}$, the maximum likelihood estimates for the parameters of the MVG are:

$$\hat{\mu} = \frac{1}{N}\sum_i \mathbf{x}^{(i)}$$

$$\hat{\Sigma} = \frac{1}{N}\sum_i (\mathrm{x}^{(\mathrm{i})} - \hat{\mu})(\mathrm{x}^{(\mathrm{i})} - \hat{\mu})^T$$

Putting it all together in LDA

- Also called Gaussian Discriminant Analysis
- Here
  - $t \sim$ Bernoulli($\mathbf{w}$)
  - $\mathbf{x}|t=0 \sim \mathrm{N}(\mu_1, \Sigma)$
  - $\mathbf{x}|t=1 \sim \mathrm{N}(\mu_2, \Sigma)$
- Writing this out, we get:

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

$$p(x \mid t=1) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)\right]$$

Picking A Class

- We again use Bayes rule:

MVG conditional feature probability

Prior class probability

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

Posterior label probability

Prior feature probability (ignored)

The Beauty of Homoscedasticity

- Recall we assumed Σ same for all classes
- When is P(y0|x)>P(y1|x)???

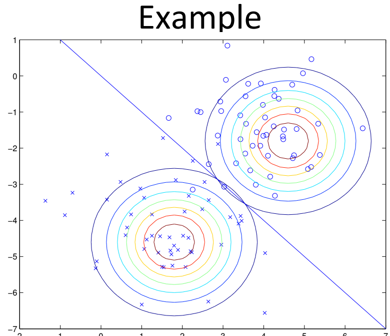
$$\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_0)^T \Sigma^{-1}(x-\mu_0)\right] p(y0) >$$

$$\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_1)^T \Sigma^{-1}(x-\mu_1)\right] p(y1)$$

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

Linear!!!

Example



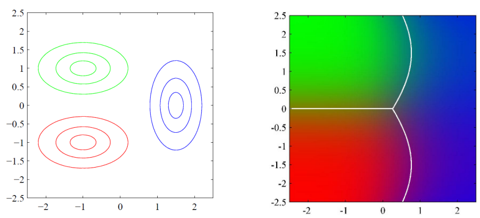
The decision boundary is at p(y=1|x) = 0.5

Homoscedastic LDA Discussion

- For multiclass, this gives convex decision boundaries
- Satisfies desiderata for multiclass decision boundaries

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

Heteroscedastic Distributions



(assuming uniform class priors, in this example)

Comparing LTU, LR, LDA

- Big debate about the relative merits of
  - direct classifiers (like LTU) versus
  - conditional models (like LR) versus
  - generative models (like LDA)

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 μ_1, μ_2, and Σ!
  - 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, logistic 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:** The 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.