Linear Classification

Ron Parr
CPS 570

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?

With content adapted from Andrew Ng, Lise Getoor, and Tom Dietterich
Figures from textbook courtesy of Chris Bishop and © Chris Bishop
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):
  \[ y(x) = \begin{cases} 
  1 & \text{if } w_1 x_1 + \cdots + w_n x_n \geq w_0 \\
  0 & \text{otherwise} 
\end{cases} \]

- 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 Least Mean Square (LMS) algorithm
  - Learn the conditional distribution: Logistic regression
  - Learn the joint distribution:
    - Naive 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:
    - $x_1^\land x_2^\land x_4^$: $1\cdot x_1 + 0\cdot x_2 + 1\cdot x_3 + 1\cdot x_4 \geq 3$
    - $x_1^\land \neg x_3^\land x_4^$: $1\cdot x_1 + 0\cdot x_2 + 1\cdot x_3 + 1\cdot x_4 \geq 2$
  - 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:
      - $(x_1^\land x_3) + (x_3^\land x_4)$
    - Exclusive-Or
      - $(x_1^\land \neg x_2) + (\neg x_1^\land x_2)$
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 $w_j^T x > w_i^T x$ 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

h is a simple step function (sgn)

\[ y = \text{sgn}(w^T x) \]

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 = \( \text{sgn}(w^T x) \)
• Perceptron_error\( (x^{(i)}, w) = -\text{sgn}(w^T x) \ast t^{(i)} \)
  – +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 : \text{\( w_j \leftarrow w_j + \alpha x_j^{(i)} t^{(i)} \)}
\]

“Learning Rate”
(can be any constant)

• \( i \) iterates over samples
• \( j \) iterates over weights
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 $p_t(x; w)$ 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
  $$p_1(x; w) = \frac{e^{w^T x}}{1 + e^{w^T x}} = \frac{1}{1 + e^{-w^T x}}$$
  $$p_0(x; w) = 1 - p_1(x; w) = \frac{e^{-w^T x}}{1 + e^{-w^T x}}$$
- This is equivalent to
  $$\log \frac{p_1(x; w)}{p_0(x; w)} = w^T x$$
- 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 \((-\infty, +\infty)\) to be positive and sum to 1 so that it can represent a probability

![Graph showing the function \(\frac{e^{w'x}}{1 + e^{w'x}}\)]

**Constructing a Learning Algorithm**

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

\[
\arg\max_{h_w} P(h_w \mid X) = \arg\max_{h_w} \frac{P(X \mid h_w)P(h_w)}{P(X)} \quad \text{by Bayes' Rule}
\]

\[
= \arg\max_{h_w} P(X \mid h_w)P(h_w) \quad \text{because } P(X) \text{ doesn't depend on } h
\]

\[
= \arg\max_{h_w} P(X \mid h_w) \quad \text{if we assume } P(h) \text{ is uniform}
\]

\[
= \arg\max_{h_w} \log P(X \mid h_w) \quad \text{because log is monotone}
\]

- The **likelihood function** views \(P(X \mid h_w)\) 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
(Not Presented, but 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 from

• Take an example \( (x^{(i)}, t^{(i)}) \)
  – if \( y^{(i)} = 0 \), the log likelihood is \( \log(1-p_{(i)}(x; w)) \)
  – if \( y^{(i)} = 1 \), the log likelihood is \( \log p_{(i)}(x; w) \)

• These two are mutually exclusive, so we can combine them to get:

\[
L(w; x^{(i)}, t) = \log P(t^{(i)} | x^{(i)}, w) = (1 - t^{(i)})\log[1 - p_{(i)}(x^{(i)}; w)] + t^{(i)}\log p_{(i)}(x^{(i)}; w)
\]

• The goal of our learning algorithm will be to find \( w \) to maximize:

\[
L(w; X, t)
\]
Computing the Gradient

\[
\frac{\partial L(w)}{\partial w_j} = \sum \frac{\partial}{\partial w_j} L(w; t^{(i)}, x^{(i)})
\]

\[
\frac{\partial}{\partial w_j} L(w; t^{(i)}, x^{(i)}) = \frac{\partial}{\partial w_j} \left( (1-t^{(i)}) \log [1-p_i(x^{(i)}; w)] + t^{(i)} \log p_i(x^{(i)}; w) \right)
\]

\[
= (1-t^{(i)}) \frac{1}{1-p_i(x^{(i)}; w)} \frac{\partial p_i(x^{(i)}; w)}{\partial w_j} + t^{(i)} \frac{1}{p_i(x^{(i)}; w)} \frac{\partial p_i(x^{(i)}; w)}{\partial w_j}
\]

\[
= \left[ t^{(i)} - \frac{p_i(x^{(i)}; w)}{1-p_i(x^{(i)}; w)} \right] \frac{\partial p_i(x^{(i)}; w)}{\partial w_j}
\]

\[
= \left[ t^{(i)} - \frac{p_i(x^{(i)}; w)}{1-p_i(x^{(i)}; w)} \right] \frac{\partial p_i(x^{(i)}; w)}{\partial w_j}
\]

\[
\frac{\partial p_i(x^{(i)}; w)}{\partial w_j} = \frac{1}{1+e^{-w^T x^{(i)}}} \frac{\partial}{\partial w_j} \left( 1+e^{-w^T x^{(i)}} \right)
\]

\[
= \frac{1}{1+e^{-w^T x^{(i)}}} e^{-w^T x^{(i)}} \frac{\partial}{\partial w_j} (w^T x^{(i)})
\]

\[
= \frac{1}{1+e^{-w^T x^{(i)}}} e^{-w^T x^{(i)}} x^{(i)}
\]

\[
= p_i(x^{(i)}; w) (1 - p_i(x^{(i)}; w)) x^{(i)}
\]

Recall: \( p_i(x; w) = \frac{e^{w^T x}}{1+e^{w^T x}} = \frac{1}{1+e^{-w^T x}} \).
Gradient cont.

- The gradient of the log likelihood for a single point is thus:
  \[
  \frac{\partial}{\partial w_j} L(w; x^{(i)}, t^{(i)}) = \frac{t^{(i)} - p_1(x^{(i)}; w)}{p_1(x^{(i)}; w) + (1 - p_1(x^{(i)}; w))} \left( \frac{\partial p_1(x^{(i)}; w)}{\partial w_j} \right) \\
  = \frac{t^{(i)} - p_1(x^{(i)}; w)}{p_1(x^{(i)}; w) + (1 - p_1(x^{(i)}; w))} p_1(x^{(i)}; w) (1 - p_1(x^{(i)}; w)) x_j^{(i)} \\
  = (t^{(i)} - p_1(x^{(i)}; w)) x_j^{(i)}
  \]

- The overall gradient is:
  \[
  \frac{\partial L(w)}{\partial w_j} = \sum_i (t^{(i)} - p_1(x^{(i)}; w)) x_j^{(i)}
  \]

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
  
  \[
  w_j^{(t+1)} = w_j^{(t)} + \alpha \sum_{i=1}^{N} (t^{(i)} - p(x^{(i)}; w^{(t)})) x_j^{(i)}
  \]
  t++

- Perceptron:
  Repeat until convergence (for every j
  
  \[
  w_j^{(t+1)} = w_j^{(t)} + \alpha \sum_{i \in \text{misclassified}} t^{(i)} x_j^{(i)}
  \]
  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.
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:

\[
\log \frac{P(y = 1 \mid x)}{P(y = K \mid x)} = \theta_1 \cdot x
\]

\[
\log \frac{P(y = 2 \mid x)}{P(y = K \mid x)} = \theta_2 \cdot x
\]

\[
\vdots
\]

\[
\log \frac{P(y = k - 1 \mid x)}{P(y = K \mid x)} = \theta_{k-1} \cdot x
\]

- The conditional probability for class k ≠ K is

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

- and for class k = K:

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

Summary of Logistic Regression

- Learns the Conditional Probability Distribution P(t | x)
- No closed form solution
- Very simple expression for gradient permits local search
  - Begin 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
  – Example: Perceptron

• Logistic Regression
  – Maximizes log likelihood

\[ \log \frac{p_1(x; w)}{p_0(x; w)} = w^T x \]

Naïve Bayes is a linear method!

• Choose class 1 when:

\[
P(x_1, \ldots, x_n \mid t_1)P(t_1) > P(x_1, \ldots, x_n \mid t_0)P(t_0)
\]

\[
P(t_1) \prod_{i=1}^{n} P(x_i \mid t_1) > P(t_0) \prod_{i=1}^{n} P(x_i \mid t_0)
\]

\[
\log(P(t_1)) + \sum_{i=1}^{n} \log(P(x_i \mid t_1)) > \log(P(t_0)) + \sum_{i=1}^{n} \log(P(x_i \mid t_0))
\]

• Fundamentally same expressive power as other linear methods
Linear Discriminant Analysis

- In LDA, we learn the distribution $P(x \mid t)$

- We assume that $x$ is continuous
- We assume $P(x \mid t)$ 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 $\{x_1, \ldots, x_N\}$, the maximum likelihood estimates for the parameters 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
  - $t \sim \text{Bernoulli}(w)$
  - $x \mid t=0 \sim \mathcal{N}(\mu_0, \Sigma)$
  - $x \mid t=1 \sim \mathcal{N}(\mu_1, \Sigma)$

Huge assumption here: Same $\Sigma$ for both classes

- Writing this out, we get:

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

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

Called the *Class Conditional* densities

Picking A Class

- We again use Bayes rule:

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

- Prior class probability
- Prior feature probability (ignored)
- Posterior label probability
- MVG conditional feature probability
The Beauty of Homoscedasticity

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

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

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

Linear!!!
Homoscedastic LDA Discussion

- For multiclass, this gives convex decision boundaries
- This is nice because it makes classification easy (easy to use geometric data structures)

- 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, 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 $\Sigma$
  - 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 $x^{(i)}$ 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 fast
- LR, NB gives probabilities, are more robust
- LDA models the data