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?

What is a Linear Discriminant?

- Simplest kind of classifier, a linear threshold unit (LTU):
  \[ y(x) = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x} \geq w_0 \\ 0 & \text{otherwise} \end{cases} \]

- We sometimes assume \( w_0 = 1 \), so \( y(x) = \mathbf{w}^T \mathbf{x} \)
- A linear discriminant is an \( n-1 \) dimensional hyperplane
- \( \mathbf{w} \) 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:
    - Naïve Bayes
    - Linear discriminant analysis (LDA)

Representing Classes

- Interpret \( \mathbf{t}^{(i)} \) as the probability that the \( i \)th element is in a particular class
- Classes usually disjoint
- For multiclass, \( \mathbf{t}^{(i)} \) may be a vector
- \( \mathbf{t}^{(i)}[j] = \mathbf{t}^{(i)} \equiv 1 \) if \( i \)th element is in class \( j \), 0 OTW
- Notation: For convenience, we will sometimes refer to the “raw” variables \( \mathbf{x} \), rather than the features as seen through the lens of our features, \( \phi \)
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:
    - $x_1 \land x_2 \land x_4$: $1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 3$
    - $x_1 \land \neg x_2 \land x_3$: $1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 2$
  - at least m-of-n
    - $1 \cdot x_1 + 1 \cdot x_2 + \ldots + 1 \cdot x_n \geq 2$

- Examples of things that cannot be expressed:
  - Non-trivial disjunctions:
    - $(x_1 \land x_3) \lor (x_3 \land x_4)$
  - Exclusive-Or
    - $(x_1 \land \neg x_2) \lor (\neg x_1 \land x_2)$

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

Perceptron Learning

• We are given a set of inputs $x^{(1)}...x^{(n)}$
• $t^{(1)}...t^{(n)}$ 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($x^{(i)}, w$) = $-\text{sgn}(w^T x) * t^{(i)}$
  – +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:

$$\forall i \in \text{misclassified} \quad \forall j: w_j \leftarrow w_j + \alpha \cdot t^{(i)} x^{(i)}$$

“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 \(P(t|\mathbf{x})\)
- Let \(p_t(\mathbf{x}; \mathbf{W})\) be our estimate of \(P(t|\mathbf{x})\), where \(\mathbf{W}\) is a vector of adjustable parameters.
- Assume there are two classes, \(t = 0\) and \(t = 1\) and
  
  \[
  p_t(\mathbf{x}; \mathbf{W}) = \frac{e^{\mathbf{w}^T \mathbf{x}}}{1 + e^{\mathbf{w}^T \mathbf{x}}}
  \]
  
  \[
  p_{\bar{t}}(\mathbf{x}; \mathbf{W}) = 1 - p_t(\mathbf{x}; \mathbf{W}) = \frac{1}{1 + e^{\mathbf{w}^T \mathbf{x}}}
  \]
- This is equivalent to
  
  \[
  \log \frac{p_t(\mathbf{x}; \mathbf{W})}{p_{\bar{t}}(\mathbf{x}; \mathbf{W})} = \mathbf{w}^T \mathbf{x}
  \]
- IOW, the log odds of class 1 is a linear function of \(\mathbf{x}\)

### 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

<table>
<thead>
<tr>
<th>(e^{\mathbf{w}^T \mathbf{x}})</th>
<th>0.0</th>
<th>0.0</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\frac{e^{\mathbf{w}^T \mathbf{x}}}{1 + e^{\mathbf{w}^T \mathbf{x}}})</td>
<td>0.0</td>
<td>10.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Constructing a Learning Algorithm

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

\[
\text{argmax}_{h} P(h | X) = \text{argmax}_{h} \frac{P(X | h) P(h)}{P(X)} \quad \text{by Bayes' Rule}
\]

\[
= \text{argmax}_{h} P(X | h) P(h) \quad \text{because } P(X) \text{ doesn't depend on } h
\]

\[
= \text{argmax}_{h} P(X | h) \quad \text{if we assume } P(h) \text{ is uniform}
\]

\[
= \text{argmax}_{h} \log P(X | h) \quad \text{because log is monotone}
\]

- 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 Gradient

\[
\frac{\partial L(w)}{\partial w_j} = \frac{\partial}{\partial w_j} \sum_{i=1}^{\hat{t}} (1 - \hat{t}_i) \log(1 - p_i(x_i^j; w)) + \hat{t}_i \log p_i(x_i^j; w)
\]

\[
= (1 - \hat{t}_i) \frac{1}{\frac{\partial}{\partial w_j} \log(1 - p_i(x_i^j; w))} + \hat{t}_i \frac{1}{\frac{\partial}{\partial w_j} \log p_i(x_i^j; w)}
\]

\[
= \left[ \frac{1}{1 - p_i(x_i^j; w)} \right] \frac{\partial p_i(x_i^j; w)}{\partial w_j}
\]

Log Likelihood for Conditional Probability Estimators

- We can express the log likelihood in a compact form.

\[
L(w; x^{(i)}, t) = \log P(t^1 | x^{(i)}, w) = (1 - t^1) \log(1 - p_1(x_i; w)) + t^1 \log p_1(x_i; w)
\]

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

\[
L(w; X)
\]

Gradient cont.

- Recall the form of \( p_i \):

\[
p_i(x_i^j; w) = \frac{e^{w^j x_i^j}}{1 + e^{w^j x_i^j}} = \frac{1}{1 + e^{-w^j x_i^j}}
\]

- So we get:

\[
\frac{\partial p_i(x_i^j; w)}{\partial w_j} = \frac{1}{(1 + e^{-w^j x_i^j})^2} e^{-w^j x_i^j} \frac{\partial}{\partial w_j} (w^j x_i^j)
\]

\[
= \frac{1}{(1 + e^{-w^j x_i^j})^2} e^{-w^j x_i^j} (x_i^j)
\]

\[
= p_i(x_i^j; w)(1 - p_i(x_i^j; w)) x_i^j
\]

Recall: \( p_i(x_i) = 1 - p_i(x_i) = \frac{e^{w^j x_i^j}}{1 + e^{w^j x_i^j}} = \frac{1}{1 + e^{-w^j x_i^j}} \)
Gradient cont.

- The gradient of the log likelihood for a single point is thus:

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

\[
= \left[ \frac{t^{(i)} - p_i(x^{(i)}; w)}{p_i(x^{(i)}; w)(1 - p_i(x^{(i)}; w))} \right] \quad \text{for } j = 1..p, i = 1..N
\]

- The overall gradient is:

\[
\frac{\partial L(w)}{\partial w_j} = \sum_i (t^{(i)} - p_i(x^{(i)}; w)) x_j^{(i)}
\]

Compare w/Perceptron rule!

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:

\[
\log \frac{P(y \neq K | x)}{P(y = K | x)} = \theta_k \cdot x
\]

- The conditional probability for class k ≠ 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:

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

Batch Ascent/Descent

- Logistic regression w/ training set \(\{(x^i, t^i), i = 1..N\}\)

Repeat until convergence:

for every j

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

\(t++\)

- Perceptron:

Repeat until convergence (for even j)

\[
w_j^{(t+1)} = w_j^{(t)} + \alpha \sum_{i \text{unclassified}} (t^i x_j^{(i)})
\]

\(t++\)

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

Naïve Bayes is a linear method!

• Choose class 1 when:

  \[
  P(x_1, ..., x_n | t_1)p(t_1) > P(x_1, ..., x_n | t_0)p(t_0)
  \]

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

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

  • Fundamentally same expressive power as other linear methods

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

• Nota bene: LDA can also mean “Latent Dirichlet Allocation”, which is something different

Estimating the MVG parameters

• Given a set of data points \( \{x_1, ..., x_n\} \), the maximum likelihood estimates for the parameters of the MVG are:

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

  \[
  \hat{\Sigma} = \frac{1}{N-1} \sum_{i=1}^{N} (x^{(i)} - \hat{\mu})(x^{(i)} - \hat{\mu})^T
  \]
Putting it all together in LDA
- Also called Gaussian Discriminant Analysis
- Here
  - \( t \sim \text{Bernoulli}(w) \)
  - \( x|t=0 \sim \mathcal{N}(\mu_0, \Sigma) \)
  - \( x|t=1 \sim \mathcal{N}(\mu_1, \Sigma) \)
- Writing this out, we get:
  \[
  p(x | t = 0) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[ -\frac{1}{2} (x - \mu_0)^\top \Sigma^{-1} (x - \mu_0) \right]
  \]
  \[
  p(x | t = 1) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[ -\frac{1}{2} (x - \mu_1)^\top \Sigma^{-1} (x - \mu_1) \right]
  \]
  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)
  MVG conditional feature probability
  Posterior label probability

The Beauty of Homoscedasticity
- Recall we assumed \( \Sigma \) same for all classes
- When is \( P(t0 | x) > P(t1 | x) \)???
  \[
  \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[ -\frac{1}{2} (x - \mu_0)^\top \Sigma^{-1} (x - \mu_0) \right] \rho(t0) > \]
  \[
  \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left[ -\frac{1}{2} (x - \mu_1)^\top \Sigma^{-1} (x - \mu_1) \right] \rho(t1)
  \]
  \[
  -(x - \mu_0)^\top \Sigma^{-1} (x - \mu_0) + k_0 > -(x - \mu_1)^\top \Sigma^{-1} (x - \mu_1) + k_b
  \]
  Linear!!!

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

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 $x^i_j$ 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 fast**
- **LR, NB gives probabilities, are more robust**
- **LDA models the data**