LINEAR MODELS

CLASSIFICATION

Continuous metric feature space
Classes of linear methods for classification
Model the boundaries between classes

Shai Fine
Agenda

- Continuous metric feature space
- Classes of linear methods for classification
- Model the boundaries between classes
  - Discriminant functions
    - Linear, Generalized Linear
  - Rosenblatt’s Perceptron learning algorithm
  - Widrow-Hoff Least Mean Square (LMS) algorithm
  - MSE and Closed-form solution
Continuous Instance Space

• In this lecture we will study the setting where all the features have continuous real values
• Assuming $d$ features, the instance space can be described as $R^d$
  • Any instance is just a points in this space
  • $R^d$ is a metric space
    • There is a way to measure distance between points in this space
• Many classification problems can be converted to a classification between points in $R^d$
  • We can base our classification on metrics!
Linear Methods for Classification

• What are they?
  • Methods that give linear decision boundaries between classes
    • Linear decision boundaries:  $\{x | w^T x + w_0 = 0\}$

• How to define decision boundaries?
  • Two classes of methods
    • Model the boundaries between classes as linear
      • Rosenblatt’s Perceptron learning algorithm
      • Widrow-Hoff Least Mean Square (LMS) algorithm
    • Model a discriminant function for each class as linear
      • Linear regression fit to the class indicator variables
      • Linear discriminant analysis (LDA)
      • Logistic regression
Modeling the Boundaries Between Classes

- Discriminant function
  \[ f(x_1, x_2, \ldots, x_d): \mathbb{R}^d \rightarrow \mathbb{R} \]
  Defines a (hyper) surface \( f(x) = 0 \) that separates \( \mathbb{R}^d \) into regions, where \( f(x) > 0 \) and \( f(x) < 0 \)
- The nuclei \( f(x) = 0 \) is called the *decision surface*

- Classification is done as follows
  - Given an instance \( x \), check if \( f(x) > 0 \) or \( f(x) < 0 \) and classify
Linear Discriminant Function

• Linear discriminant:

\[ g(x|w, w_0) = w^T x + w_0 = \sum_{j=1}^{d} w_j x_j + w_0 \]

  - Where \( w \) is called the \textit{weight vector} and \( w_0 \) the \textit{bias or threshold weight}

• Linear discriminant function defines a hyper-plane \( g(x|w, w_0) = 0 \) in \( R^d \)
  - \( w \) is normal to any vector lying in the hyperplane

• If we define the \((d + 1)\) dimensional vectors

  \( \tilde{x} = (x, 1) \) and \( \tilde{w} = (w, w_0) \)

  then we can rewrite the linear discriminant function

  \[ g(\tilde{x}|\tilde{w}) = \tilde{w}^T \tilde{x} \]

  - With this notation, which will often prove convenient, we can interpreted \( g(\tilde{x}|\tilde{w}) = 0 \) as a \( d \) dimensional hyper-plain that pass through the origin in \( R^{d+1} \)

• Advantages
  • Simple
    • O(d) space/computation
  • Knowledge extraction
    • Weighted sum of attributes – positive/negative weights, magnitudes
Two Classes in a Plane

- When a new instance is given
  - We first place the new instance in the space
  - Classify it according to the subspace in which it resides

\[ g(x) = w_1 x_1 + w_2 x_2 + w_0 = 0 \]

\[ g(x) < 0 \]
\[ g(x) > 0 \]

\[ \text{choose } \begin{cases} C_1 & \text{if } g(x) > 0 \\ C_2 & \text{otherwise} \end{cases} \]
The Perceptron

- A Perceptron is a simple model of a human neuron
  - weights = “synapses”
  - threshold = “neuron firing”

\[
\sum_{i=0}^{n} w_i x_i
\]

A Perceptron calculates
- A weighted sum of the input features
- This sum is then thresholded
  - produces +1 or -1
- Instead of a threshold, a “sigmoid” function (“soft threshold”) can be used

\[
s(f) = \begin{cases} 
1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\
-1 & \text{otherwise}
\end{cases}
\]
Prehistory

- This seminal paper pointed out that simple artificial “neurons” could be made to perform basic logical operations such as AND, OR and NOT

**Truth Table for Logical AND**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x &amp; y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Perceptron Expressive Power

• What mappings can a Perceptron represent perfectly?
  • A perceptron is a linear classifier
  • Thus it can represent any mapping that is linearly separable
  • some Boolean functions like AND and OR (on left)
  • but not Boolean functions like XOR (on right)
Rosenblatt’s Perceptron

- Subsequent progress was inspired by the invention of learning rules inspired by ideas from neuroscience…
- Rosenblatt’s Perceptron could automatically learn to categorize or classify input vectors into types

It obeyed the following rule
If the sum of the weighted inputs exceeds a threshold, output 1, else output -1.

\[
1 \text{ if } \sum \text{input}_i \times \text{weight}_i > \text{threshold} \\
-1 \text{ if } \sum \text{input}_i \times \text{weight}_i < \text{threshold}
\]
Linear Classification

Perceptron Learning

Algorithm 1 Perceptron Rule

1: $\bar{w} \leftarrow \bar{w}_0$
2: ErrorFound $\leftarrow$ True
3: while ErrorFound do
4:   ErrorFound $\leftarrow$ False
5:   for $i = 1, \ldots, N$ do
6:     $\hat{y}_i \leftarrow \text{sgn}(\bar{x}_i \cdot \bar{w}_t)$
7:     if $\hat{y}_i \neq y_i$ then
8:       ErrorFound $\leftarrow$ True
9:       $\bar{w} \leftarrow \bar{w} + \eta y_i \bar{x}_i$
10:    end if
11:   end for
12: end while
13: return $\bar{w}$

The Perceptron learning rule (Rosenblatt, ’58)
“Oldest trick in the book”, Seriously …

When a new instance is given
Classify according to the separating hyperplane
The Perceptron Criterion

• The goal in learning a perceptron – Minimize classification Error
  • Adjust the weights of the perceptron to provide the correct output for each sample

• The error is piecewise constant
  • Instead, work with a piecewise linear called the Perceptron Criterion

• The Perceptron Criterion

\[ E_{perc}(w) = - \sum_{i \in D_{miss}} w^T (x^i y^i) \]

  • Where \( D_{miss} \) is the set of sample points which are misclassified by \( w \), and \( t^i \)'s are the true classifications (targets) of the \( x^i \) points

• Properties of the error function \( E_{perc}(w) \)
  • It is a sum of positive terms, and equals zero if all the points are correctly classified
  • A sum, over all misclassified points, of the absolute distance to the decision boundary
Rosenblatt’s Perceptron Learning Algorithm

- Finds a separating hyper-plane by minimizing the distance of misclassified points to the decision boundary
  - I.e. Minimize: \( E^{perc}(w) = -\sum_{i \in D_{miss}} w^T(x^i y^i) \)

- The gradient of the error
  \[
  \nabla_w E^{perc} = \left[ \frac{\partial E^{perc}(w)}{\partial w_1}, \frac{\partial E^{perc}(w)}{\partial w_2}, \ldots, \frac{\partial E^{perc}(w)}{\partial w_d} \right]
  \]
  - Where \( \frac{\partial E^{perc}(w)}{\partial w_j} = -\sum_{i \in D_{miss}} x_j^i y^i \)

- Rosenblatt’s Learning Algorithm is \textit{Stochastic Gradient Descent}
  - The training set is visited in some order and for each misclassified instance the weights are updated in the negative direction of the gradient
    \[
    w^{new} = w + \eta \Delta w = w - \eta \nabla_w E^{perc}
    \]
  - Equivalently, the \textit{Perceptron Learning Rule} for the \( j \)th weight
    \[
    w_j^{new} = w_j + \eta \Delta w_j = w_j + \eta (y^i - o^i)x_j^i
    \]
    - \( o^i \) is the Perceptron’s classification (output) of the \( i \)th instance
    - \( \eta \) is a small constant called the learning rate
  - If \( y^i = o^i \), the Perceptron is correct and there is no update

- If training data is \textit{linearly separable} It can be shown that the Perceptron algorithm converges to a separating hyper-plane in a finite number of steps
Perceptron Mistake Bound Theorem

Novikoff, 1962

Let \((x_1, y_1), \ldots, (x_n, y_n)\), where \(x_i \in \mathbb{R}^N\) and \(y_i \in \{-1,1\}\) be a sequence of labeled example and assume that it is separable, and let \(R = \max_i \|x_i\|\).

Suppose that there exist a vector \(w_{opt} \in \mathbb{R}^N\) and \(\gamma > 0\) such that \(\|w_{opt}\| = 1\) and \(\forall i, \ y_i(w_{opt}^T x_i) \geq \gamma\).

Then, the number of mistakes made by the Perceptron algorithm on this sequence of examples is at most

\[
\left(\frac{R}{\gamma}\right)^2
\]

The bound depends only on the normalized margin \(\frac{\gamma}{R}\), thus it is invariant to scaling, and independent of the learning rate \(\eta\) and the dimension\(^*\).

\(^*\) there is a connection through the parameter \(R\)
Perceptron Mistake Bound Theorem

Proof sketch

• Let \( w_0 = 0 \) be the initial weight vector, and denote \( w_k \) the weight vector after the \( k \)-th mistake. Assume that the \( k \)-th mistake occurs on the \( i \)-th example \((x_i, y_i)\), namely \( y_i(w_k^T x_i) \leq 0 \)

  • Then
    • \( w_{k+1} = w_k + \eta y_i x_i \)  
    • \( w_{opt}^T w_{k+1} = w_{opt}^T w_k + \eta y_i (w_{opt}^T x_i) \)  
    • \( w_{opt}^T w_{k+1} \geq w_{opt}^T w_k + \eta \gamma \)  
    • \( w_{opt}^T w_{k+1} \geq k \eta \gamma \)  

  • On the other hand
    • \( \|w_{k+1}\|^2 = \|w_k\|^2 + 2\eta y_i (w_k^T x_i) + \eta^2 \|x_i\|^2 \)  
    • \( \|w_{k+1}\|^2 \leq \|w_k\|^2 + \eta^2 R^2 \)  
    • \( \|w_{k+1}\|^2 \leq k \eta^2 R^2 \)  

  • Therefore
    • \( \|w_{opt}\| \sqrt{k\eta R} \geq \|w_{opt}\| \|w_{k+1}\| \geq w_{opt}^T w_{k+1} \geq k \eta \gamma \)  
    • \( k \leq R^2 \gamma^2 \)  

The perceptron rule

Multiply by \( w_{opt}^T \)

Since \( \forall i, \ y_i (w_{opt}^T x_i) \geq \gamma \)

By induction

\( w_k \) makes a mistake on \((x_i, y_i)\)

By induction

Cauchy-Schwartz inequality

Since \( \|w_{opt}\| = 1 \)
Least Mean Square (LMS) Algorithm

*Widrow-Hoff Alg.*

- **ADALINE** (Adaptive Linear Neuron) network and its learning rule, **LMS** (Least Mean Square) were proposed by *Widrow and Hoff* in 1960
  - The LMS algorithm minimizes *mean square error (MSE)*
    \[
    E(w) = \frac{1}{2} \sum_{i \in D} (y^i - w^T x^i)^2
    \]
    - Where \(D\) is the training set
  - The gradient of the error
    \[
    \frac{\partial E}{\partial w_j} = \frac{1}{2} \sum_{i \in D} 2(y^i - w^T x^i) \frac{\partial}{\partial w_j} (y^i - w^T x^i) = \sum_{i \in D} (y^i - w^T x^i)(-x_j^i)
    \]
  - The *Widrow-Hoff Learning Rule* for the \(j^{th}\) weight
    \[
    w_j^{new} = w_j + \eta \Delta w_j = w_j + \eta (y^i - w^T x^i)x_j^i
    \]
    - Recall that for the Perceptron learning rule – \(o^i = \text{sign}(w^T x^i)\)

- Both LMS and the Perceptron algorithms suffer from the same limitation
  - They can only solve linearly separable problems
LMS Problem

Minimize the distance between the positive instances and the +1 iso-line of the function

Minimize the distance between the negative instances and the -1 iso-line of the function

\[ E[w] \equiv \frac{1}{2} \sum_{i \in \mathcal{D}} (w^T x^i - y^i)^2 = \frac{1}{2} \sum_{i \in \mathcal{D}^+} (w^T x^i - 1)^2 + \frac{1}{2} \sum_{i \in \mathcal{D}^-} (w^T x^i + 1)^2 \]
Non Iterative Solution?

• Both LMS and Perceptron algorithms loop through the training samples several times until convergence
  • in stochastic even one at a time

• Compare to decision trees
  • no meaning to loop through training samples

• Can there be a different way to solve this directly?
Closed-form Solution

\[ X \vec{w} = \vec{y} \]

\[
\begin{pmatrix}
    x_0^1 & \cdots & x_d^1 \\
    \vdots & \ddots & \vdots \\
    x_0^n & \cdots & x_d^n
\end{pmatrix}
\begin{pmatrix}
    w_1 \\
    \vdots \\
    w_d
\end{pmatrix}
= 
\begin{pmatrix}
    y_1 \\
    \vdots \\
    y_m
\end{pmatrix}
\]

- If \( X \) is non singular we can write \( \vec{w} = X^{-1}\vec{y} \)
- Most of the time, \( X \) is singular (e.g. \( n > d \))
Back to Minimum Squared Error

• Lets look at the error vector \( e = X \overrightarrow{w} - \overrightarrow{y} \)

• Minimizing the squared length of this error vector is equivalent to minimizing the sum of squared distance function

\[
E(\overrightarrow{w}) = \| X \overrightarrow{w} - \overrightarrow{y} \|^2 = \sum_{i=1}^{n} (w^T x^i - y^i)^2
\]

• Forming the derivative yields

\[
\nabla E(\overrightarrow{w}) = 2X^T (X \overrightarrow{w} - \overrightarrow{y})
\]
Pseudo-inverse

• Setting the derivative to zero yields the necessary condition for minimum

\[ X^T X \bar{w} = X^T \bar{y} \]

• Now, \( X^T X \) is square and often nonsingular and so we can solve for \( \bar{w} \) uniquely as

\[ \bar{w} = (X^T X)^{-1} X^T \bar{y} \]

• The \( d \times d \) matrix \( (X^T X)^{-1} X^T \) is called the *Pseudo Inverse* of \( X \)
  • If \( X \) is square, then \( (X^T X)^{-1} X^T = X^{-1} \)
    • i.e. just its inverse

• Difficulties with Pseudo Inverse
  • Must work with very large matrices
    • Number of instances can be hundreds and number of features tens or more
  • \( X^T X \) can still be singular and not have an inverse!

\( (*) \) Moore–Penrose inverse
The Fall of the Perceptron

  - Before too long researchers had begun to discover the Perceptron’s limitations
    - Unless input categories were “linearly separable”, a perceptron could not learn to discriminate between them
    - Unfortunately, it appeared that many important categories were not linearly separable

- There is no doubt that Minsky and Papert's book was a block to the funding of research in neural networks for more than ten years
  - The book was widely interpreted as showing that neural networks are basically limited and fatally flawed

- The next progress
  - Multilayer Perceptron (using linear transfer functions)
  - Artificial Neural Network (using general transfer functions)
Summary

• Perceptron training rule guaranteed to succeed if
  • Training examples are linearly separable
  • In the in separable case
    • Sufficiently small learning rate $\eta$

• Limited representation power
  • Use networks of Perceptrons
  • Use generalized linear discriminants

• Both LMS and Perceptron algorithms
  • Use gradient descent rules

• There exist a closed-form solution to the MSE using Pseudo Inverse
BACKUP
Geometry

Reminder: A distance of a point from a line
\[ \frac{|w_1 x_1 + w_2 x_2 + w_0|}{\sqrt{w_1^2 + w_2^2}} \]
The derivative of $f: \mathbb{R} \to \mathbb{R}$ is a function $f': \mathbb{R} \to \mathbb{R}$ given by:

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$
The Gradient

2-D illustration
The gradient of $f: \mathbb{R}^2 \to \mathbb{R}$ is the function $\nabla f: \mathbb{R}^2 \to \mathbb{R}^2$

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$
Optimality conditions

If $f$ has local optimum at $x^*$ then $\nabla f(x^*) = 0$

Intuition:
Gradient descent

- Start at $x_0\ k = 0$

- Compute a search direction:
  $$-\nabla f(x_k)$$

- Compute a step length $\alpha_k$, such that:
  $$f(x_k - \alpha_k\nabla f(x_k)) \leq f(x_k)$$

- Update step:
  $$x_{k+1} = x_k - \alpha_k\nabla f(x_k)$$

- Check for convergence (stopping criteria)
  $$\|\nabla f(x_{k+1})\| \leq \varepsilon$$

- Back to first step