Lecture 5

Machine Learning

Linear Classifier

Multiple Classes
• Linear Classifier
  • Multiple classes
    1. Use collection of 2-class classifiers
      • one vs. all
      • all pairs
    2. Design multi-class loss functions
      • Perceptron Loss Function
      • Softmax Loss Function
  • Weight Regularization
• Have classes 1, 2, …, m
• Can construct multi-class classifier based on 2-class classifiers
• One way
  • Assume each 2-class classifier also gives confidence
    • Distance from separating hyperplane
      • Higher distance, more confidence
  • Train m 2-class classifiers
    • 1 vs other classes
    • 2 vs. other classes
    • ….
    • m vs. other classes
    • Make sure number of examples is balanced during training
  • At test time, run new sample through m binary classifiers
    • highest confidence class “wins”
• Works for any type of 2-class classifier, not just linear
Using 2-class Case: All pairs

- Train 2-class classifier for each distinct pair of classes \((i,j)\)

- At test time, run new example \(x\) through all binary classifiers
  - Choose most frequently occurring class
  - For example, \(x\) was classified
    - 1 time as class 1
    - 2 times as class 2
    - 0 times as class 3
    - 3 times as class 4
    - decide class 4
Multiple Classes: General Case

- General multiclass case
  - not based on 2-class classifiers
- Define $m$ linear discriminant functions
  \[ g_i(x) = w_i^T x + w_{i0} \quad \text{for } i = 1, 2, \ldots, m \]
- Assign $x$ to class $i$ if
  \[ g_i(x) > g_j(x) \quad \text{for all } j \neq i \]
- Let $R_i$ be decision region for class $i$
  - all samples in $R_i$ assigned to class $i$
Multiple Classes

- Can be shown that decision regions are convex
- In particular, they must be spatially contiguous
Thus applicability of linear classifiers is limited to mostly unimodal distributions, such as Gaussian.

For not unimodal data, need non-contiguous decision regions.

Linear classifier will fail.
• Assume examples \( \mathbf{x} \) are augmented with extra feature 1, no need to write bias explicitly
  - but from now on will not change notation to \( \mathbf{z} \)'s
• Define \( m \) discriminant functions
  \[
g_i(\mathbf{x}) = \mathbf{w}_i^t \mathbf{x} \quad \text{for } i = 1, 2, \ldots, m
\]
• Assign \( \mathbf{x} \) to \( i \) that gives maximum \( g_i(\mathbf{x}) \)
• Picture illustration

\[
\begin{bmatrix}
5 \\
3 \\
-9 \\
10
\end{bmatrix}
\]
pile all outputs into one vector
decide class 4
Multiclass Linear Classifier: Matrix Notation

- Could use one dimensional output $y_i \in \{1, 2, 3, ..., m\}$
- Convenient to use multi-dimensional outputs

$$
\begin{bmatrix}
1 \\
0 \\
0 \\
0 \\
\end{bmatrix} \quad y' = \begin{bmatrix}
0 \\
1 \\
0 \\
0 \\
\end{bmatrix} \quad y' = \begin{bmatrix}
0 \\
0 \\
1 \\
0 \\
\end{bmatrix} \quad y' = \begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
\end{bmatrix}
$$

class 1  class 2  class 3  class 4

- For training, if sample is of class $i$, want output vector to be 0 everywhere except position $i$, where it should be 1

$x$ is of class 2

$g_1(x)$
$g_2(x)$
$g_3(x)$
$g_4(x)$

$$
\begin{bmatrix}
5 \\
3 \\
-9 \\
10 \\
\end{bmatrix} \quad \begin{bmatrix}
0 \\
1 \\
0 \\
0 \\
\end{bmatrix}
got this \quad want this$$
Multiclass Linear Classifier: Matrix Notation

- Assign \( x \) to \( i \) that gives maximum \( g_i(x) = w_i^T x \)

- In matrix notation

- Assign \( x \) to class that corresponds to largest row of \( Wx \)
Assign sample $x^i$ to class that corresponds to largest row of $Wx^i$

Loss function?

$$Wx^i \quad y^i$$

Can use quadratic loss per sample $x^i$ as $\frac{1}{2}||Wx^i - y^i||^2$

- for example above, loss $(2^2 + 4^2 + 47^2 + 44^2)/2$
- total loss on all training samples $L(W) = \frac{1}{2} \Sigma_i ||Wx^i - y^i||^2$
- gradient of the loss

$$\nabla L(W) = \sum_i (Wx^i - y^i)(x^i)^t$$

- $\nabla L(W)$ has the same shape as the same shape as $W$
- batch gradient descent update

$$W = W - \alpha \sum_i (Wx^i - y^i)(x^i)^t$$
Quadratic Loss Function

- Consider gradient descent update, single sample $\mathbf{x}$ with $\alpha = 1$

$$
\mathbf{W} = \mathbf{W} - (\mathbf{Wx} - \mathbf{y})\mathbf{x}^t
$$

- Suppose $\mathbf{x} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$ and is of class 2 and $\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix}$

- update rule

$$
\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ -17 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -17 \end{bmatrix}
$$
Quadratic Loss Function

\[ Wx - y = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 23 \\ -17 \end{bmatrix} \]

- With new \( W = \begin{bmatrix} 2 & 4 & -7 \\ 6 & -12 & -4 \\ -19 & -64 & -44 \\ 19 & 44 & 35 \end{bmatrix} \), \( Wx = \begin{bmatrix} 0 \\ -38 \\ -299 \\ 221 \end{bmatrix} \)

- Already saw that quadratic loss does not work that well for classification
**Perceptron Loss**

- Generalize Perceptron loss to multiclass setting
- Per-example loss: largest score minus score for the correct class

\[
\begin{bmatrix}
2 \\
-4 \\
47 \\
-43 \\
\end{bmatrix}
\begin{bmatrix}
y^i \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
20 \\
40 \\
17 \\
-43 \\
\end{bmatrix}
\begin{bmatrix}
y^i \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 \\
0 \\
1 \\
0 \\
\end{bmatrix}
\]

**Formula for Perceptron loss on sample** \( x^i \)

\[
L_i(W) = \max_k [(Wx^i)_k - (Wx^i)_c]
\]

- \((Wx^i)_k\) is the entry in row \( k \) of vector \( Wx^i \)
- \( c \) is the correct class of sample \( x^i \)
Gradient of loss on one example

- $c$ is the correct class row
- $r$ is the row where $Wx^i$ is largest
- If $r = c$, \[-\nabla L_i(W) = 0\]
- Otherwise, \[-\nabla L_i(W) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \times^i & \times^i & \times^i & \times^i \end{bmatrix}\]

Example

\[
\begin{bmatrix} 2 \\ -4 \\ 47 \\ -43 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}
\]
Perceptron Loss Function: Example Cont.

\[-\nabla L_i(W) = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
-1 & -3 & -2 \\
1 & 3 & 2 \\
\end{bmatrix} \]

\[x^i = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} \]

\[y^i = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \]

- With \(\alpha = 1\), new \(W = \begin{bmatrix}
2 & 4 & -7 \\
9 & -3 & 2 \\
4 & 5 & 2 \\
2 & -7 & 1 \\
\end{bmatrix} + \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
-1 & -3 & -2 \\
1 & 3 & 2 \\
\end{bmatrix} = \begin{bmatrix}
2 & 4 & -7 \\
9 & -3 & 2 \\
3 & 2 & 0 \\
3 & -4 & 3 \\
\end{bmatrix} \]

- With new weights

\[Wx^i = \begin{bmatrix} 0 \\ 4 \\ 9 \\ -3 \end{bmatrix} \]

- Compare to the old weights

\[W_{old}x^i = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix} \]

- \(\) ok
- too large
- too small

With new weights

\[Wx^i = \begin{bmatrix} 0 \\ 4 \\ 9 \\ -3 \end{bmatrix} \]
• Define \( \text{softmax}(\mathbf{a}) \) function

\[
\begin{bmatrix}
\frac{\exp(a_1)}{\sum_{j=1}^{4} \exp(a_j)} \\
\frac{\exp(a_2)}{\sum_{j=1}^{4} \exp(a_j)} \\
\frac{\exp(a_3)}{\sum_{j=1}^{4} \exp(a_j)} \\
\frac{\exp(a_4)}{\sum_{j=1}^{4} \exp(a_j)}
\end{bmatrix}
\]

• Example

\[
\begin{bmatrix}
-3 \\
2 \\
1
\end{bmatrix}
\]

\[
\text{softmax}
\]

\[
\frac{\exp(-3)}{\exp(-3)+\exp(2)+\exp(1)}
\]

\[
\frac{\exp(2)}{\exp(-3)+\exp(2)+\exp(1)}
\]

\[
\frac{\exp(1)}{\exp(-3)+\exp(2)+\exp(1)}
\]

\[
\begin{bmatrix}
0.005 \\
0.7275 \\
0.2676
\end{bmatrix}
\]

• Softmax renormalizes a vector so that it can be interpreted as a vector of probabilities
Softmax Loss Function

- Generalization of logistic regression to multiclass case

- Instead of raw scores

\[
\begin{bmatrix}
  w_1^T x \\
  w_2^T x \\
  w_3^T x \\
  w_4^T x
\end{bmatrix}
= \begin{bmatrix}
  2 \\
  -1 \\
  5 \\
  -3
\end{bmatrix}
\]

- Use softmax scores

\[
\text{softmax} \left( \begin{bmatrix}
  w_1^T x \\
  w_2^T x \\
  w_3^T x \\
  w_4^T x
\end{bmatrix} \right) = \text{softmax} \left( \begin{bmatrix}
  2 \\
  -1 \\
  5 \\
  -3
\end{bmatrix} \right) = \begin{bmatrix}
  0.0473 \\
  0.0024 \\
  0.9500 \\
  0.0003
\end{bmatrix} = \begin{bmatrix}
  \text{Pr(class1)} \\
  \text{Pr(class2)} \\
  \text{Pr(class3)} \\
  \text{Pr(class4)}
\end{bmatrix}
\]

- Classifier output interpreted as probability for each class
Gradient Descent: Softmax Loss Function

- Optimize under $-\log \Pr(y^i)$ loss function

- Example

\[
x^i = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}, \quad y^i = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}
\]

\[
W = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix}
\]

\[
\text{softmax}(Wx^i) = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix}
\]

\[
\begin{bmatrix}
0.0000000000102619 \\
0.0000000005602796 \\
0.99999994294585 \\
0.0000000000000001
\end{bmatrix} = 
\begin{bmatrix}
\Pr(\text{class1}) \\
\Pr(\text{class2}) \\
\Pr(\text{class3}) \\
\Pr(\text{class4})
\end{bmatrix}
\]

- Loss on this example is $-\log(0.0000000000000001) = 40$
Gradient Descent: Softmax Loss Function

- Update rule for weight matrix $W$

$$W = W + \alpha \sum_i \left( y^i - \text{softmax}(W^T x^i) \right) (x^i)^t$$

- Example, single sample gradient descent with $\alpha = 0.1$

$$x^i = \begin{bmatrix} 1 \\ 3 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad y^i = \begin{bmatrix} 2 \\ 9 \\ 4 \\ 0 \end{bmatrix}, \quad w = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix}, \quad wx^i = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix}$$

- Update for $W$

$$W = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} + 0.1 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} - \text{softmax} \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 3.9 & 4.7 & 1.8 \\ 2.1 & -6.7 & 1.2 \end{bmatrix}$$
Generalized Linear Classifier

- Can use other discriminant functions, like quadratics
  \[ g(x) = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2 \]

- Methodology is almost the same as in the linear case
  - \[ f(x) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{22} x_2^2) \]
  - \[ z = \begin{bmatrix} 1 & x_1 & x_2 & x_1 x_2 & x_1^2 & x_2^2 \end{bmatrix} \]
  - \[ a = \begin{bmatrix} w_0 & w_1 & w_2 & w_{12} & w_{11} & w_{22} \end{bmatrix} \]
  - use gradient descent to minimize Perceptron loss function, any other loss function

- Can add any degree polynomial features
Generalized Linear Classifier

- Generalized linear classifier
  \[ g(x, w) = w_0 + \sum_{i=1}^{m} w_i h_i(x) \]

- \( h(x) \) are called basis function, can be arbitrary functions
  - In strictly linear case, \( h_i(x) = x_i \)

- Linear function in its parameters \( w \)
  \[ g(x, w) = w_0 + w^t h \]
  \[ h = [h_1(x) \ h_2(x) \ \ldots \ h_m(x)] \]
  \[ [w_1 \ \ldots \ w_m] \]

- Use the same training methods as before with new feature vector \( h \)
• Usually face severe overfitting
  • too many degrees of freedom
  • boundary can “curve” to fit to the noise in the data

• Regression example
Generalized Linear Classifier

- Helps to regularize by keeping $w$ small
  - small $w$ means the boundary is not as curvy
- Regression example

![Graphs showing regression examples](image)

<table>
<thead>
<tr>
<th>Polynomial Coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M = 0$</td>
</tr>
<tr>
<td>$w_0^*$</td>
</tr>
<tr>
<td>$w_1^*$</td>
</tr>
<tr>
<td>$w_2^*$</td>
</tr>
<tr>
<td>$w_3^*$</td>
</tr>
<tr>
<td>$w_4^*$</td>
</tr>
<tr>
<td>$w_5^*$</td>
</tr>
<tr>
<td>$w_6^*$</td>
</tr>
<tr>
<td>$w_7^*$</td>
</tr>
<tr>
<td>$w_8^*$</td>
</tr>
<tr>
<td>$w_9^*$</td>
</tr>
</tbody>
</table>
• Helps to *regularize* by keeping $w$ small
  • small $w$ means the boundary is not as curvy
• For example, add $\lambda \|w\|^2$ to the loss function
• Recall quadratic loss function
  \[ L = \frac{1}{2} \sum_i \| f(x_i, w) - y_i \|^2 \]
• Regularized version
  \[ L = \frac{1}{2} \sum_i \| f(x_i, w) - y_i \|^2 + \lambda \|w\|^2 \]

• Regression example, polynomial coefficients for degree $M = 9$
• With weight regularizer, gradient of loss function has a new term $-\alpha \lambda w$

<table>
<thead>
<tr>
<th></th>
<th>small $\lambda$</th>
<th>medium $\lambda$</th>
<th>large $\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_0^*$</td>
<td>0.35</td>
<td>0.35</td>
<td>0.13</td>
</tr>
<tr>
<td>$w_1^*$</td>
<td>232.37</td>
<td>4.74</td>
<td>-0.05</td>
</tr>
<tr>
<td>$w_2^*$</td>
<td>-5321.83</td>
<td>-0.77</td>
<td>-0.06</td>
</tr>
<tr>
<td>$w_3^*$</td>
<td>48568.31</td>
<td>-31.97</td>
<td>-0.05</td>
</tr>
<tr>
<td>$w_4^*$</td>
<td>-231639.30</td>
<td>-3.89</td>
<td>-0.03</td>
</tr>
<tr>
<td>$w_5^*$</td>
<td>640042.26</td>
<td>55.28</td>
<td>-0.02</td>
</tr>
<tr>
<td>$w_6^*$</td>
<td>-1061800.52</td>
<td>41.32</td>
<td>-0.01</td>
</tr>
<tr>
<td>$w_7^*$</td>
<td>1042400.18</td>
<td>-45.95</td>
<td>-0.00</td>
</tr>
<tr>
<td>$w_8^*$</td>
<td>-557682.99</td>
<td>-91.53</td>
<td>0.00</td>
</tr>
<tr>
<td>$w_9^*$</td>
<td>125201.43</td>
<td>72.68</td>
<td>0.01</td>
</tr>
</tbody>
</table>
• $\lambda$ is a meta-parameter, cannot tune on training data
  • use validation or cross-validation to set it to a good value

• Consider polynomial of degree M=9 regression