Evaluation of classifiers
MLPs

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Evaluation

For any data set we use to test the model we can build a confusion matrix:

- Counts of examples with:
  - class label $\omega_j$ that are classified with a label $\alpha_i$

<table>
<thead>
<tr>
<th>predict $\alpha$</th>
<th>target $\omega = 1$</th>
<th>target $\omega = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 1$</td>
<td>140</td>
<td>17</td>
</tr>
<tr>
<td>$\alpha = 0$</td>
<td>20</td>
<td>54</td>
</tr>
</tbody>
</table>
### Evaluation

For any data set we use to test the model we can build a **confusion matrix**:

<table>
<thead>
<tr>
<th></th>
<th>( \omega = 1 )</th>
<th>( \omega = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict</td>
<td>( \alpha = 1 )</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>( \alpha = 0 )</td>
<td>20</td>
</tr>
</tbody>
</table>

Error: \( \frac{37}{231} \)

**Accuracy** = 1 - Error = \( \frac{194}{231} \)
Evaluation for binary classification

Entries in the confusion matrix for binary classification have names:

<table>
<thead>
<tr>
<th>predict</th>
<th>$\alpha = 1$</th>
<th>$\alpha = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega = 1$</td>
<td>$TP$</td>
<td>$FP$</td>
</tr>
<tr>
<td>$\omega = 0$</td>
<td>$FN$</td>
<td>$TN$</td>
</tr>
</tbody>
</table>

TP: True positive (hit)
FP: False positive (false alarm)
TN: True negative (correct rejection)
FN: False negative (a miss)

Additional statistics

- **Sensitivity (recall)**
  \[ SENS = \frac{TP}{TP + FN} \]

- **Specificity**
  \[ SPEC = \frac{TN}{TN + FP} \]

- **Positive predictive value (precision)**
  \[ PPT = \frac{TP}{TP + FP} \]

- **Negative predictive value**
  \[ NPV = \frac{TN}{TN + FN} \]
Binary classification: additional statistics

- Confusion matrix

<table>
<thead>
<tr>
<th>predict</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>140</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>180</td>
</tr>
</tbody>
</table>

PPV = 140/150
NPV = 180/200

SENS = 140/160
SPEC = 180/190

Row and column quantities:
- Sensitivity (SENS)
- Specificity (SPEC)
- Positive predictive value (PPV)
- Negative predictive value (NPV)

Binary decisions: ROC

- Probabilities:
  - SENS  \( p ( x > x^* \mid x \in \omega_1 ) \)
  - SPEC  \( p ( x < x^* \mid x \in \omega_0 ) \)
Receiver Operating Characteristic (ROC)

- ROC curve plots:
  \[ SN = p(x > x^* \mid x \in \omega_1) \]
  \[ 1-SP = p(x > x^* \mid x \in \omega_0) \]
  for different \( x^* \)

ROCCurve.png

Case 1  Case 2  Case 3

ROC curve
Receiver operating characteristic

• **ROC**
  – shows the discriminability between the two classes under different decision biases

• **Decision bias**
  – can be changed using different loss function

Zero-one loss function

• **Misclassification error**
  – Based on the zero-one loss function
  • Any misclassified example counts as 1
  • Correctly classified example counts as 0

<table>
<thead>
<tr>
<th>( \omega ) = 0</th>
<th>( \omega ) = 1</th>
<th>( \omega ) = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha = 0 )</td>
<td>140</td>
<td>20</td>
</tr>
<tr>
<td>( \alpha = 1 )</td>
<td>17</td>
<td>54</td>
</tr>
<tr>
<td>( \alpha = 2 )</td>
<td>12</td>
<td>4</td>
</tr>
</tbody>
</table>
General loss function

- **Error function based on a more general loss function**
  - Different misclassifications have different weight (loss)
  - \( \alpha_i \) our choice
  - \( \omega_j \) true label
  - \( \lambda(\alpha_i | \omega_j) \) loss for classification

**Example:**

\[
\begin{array}{cccc}
\alpha = 0 & \omega = 0 & \omega = 1 & \omega = 2 \\
0 & 1 & 5 \\
\alpha = 1 & 3 & 0 & 2 \\
\alpha = 2 & 3 & 1 & 0 \\
\end{array}
\]

Bayesian decision theory

- **More general loss function**
  - Different misclassifications have different weight (loss)
  \[
  \lambda(\alpha_i | \omega_j)
  \]
- **Expected loss for the classification choice** \( \alpha_i \)
  \[
  R(\alpha_i | x) = \sum_j \lambda(\alpha_i | \omega_j) P(y = \omega_j | x)
  \]
  - Also called conditional risk
- **Decision rule:** \( \alpha(x) \)
  - Chooses label (action) according to the input
- **The optimal decision rule**
  \[
  \alpha^*(x) = \arg\min_{\alpha_i} \sum_j \lambda(\alpha_i | \omega_j) P(y = \omega_j | x)
  \]
Bayesian decision theory

- The optimal decision rule
  \[ \alpha^* (x) = \arg \min_{\alpha_i} \sum_j \lambda(\alpha_i | \omega_j)P(y_j | x) \]

  How to modify classifiers to handle different loss?
  - **Discriminative models:**
    - Directly optimize the parameters according to the new loss function
  - **Generative models:**
    - Learn probabilities as before
    - Decisions about classes are biased to minimize the empirical loss (as seen above)

Calculating the loss for data

- **Confusion matrix:**
  - Counts of examples with:
  - class label \( \omega_j \) that are classified with a label \( \alpha_i \)
  - \( \begin{array}{c|ccc}
        & \omega = 0 & \omega = 1 & \omega = 2 \\
  \hline
  \alpha = 0 & 140 & 20 & 22 \\
  \alpha = 1 & 17 & 54 & 8 \\
  \alpha = 2 & 12 & 4 & 76 \\
  \end{array} \)
  - agreement

- **Loss**
  \[ \frac{1}{N} \sum_i \sum_j \lambda(\alpha_i | \omega_j)N(\alpha_j | \omega_j) \]
Multilayer neural networks

Linear units

Linear regression

\[ f(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j \]

Logistic regression

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

On-line gradient update:

\[ w_0 \leftarrow w_0 + \alpha (y - f(\mathbf{x})) \]

\[ w_j \leftarrow w_j + \alpha (y - f(\mathbf{x})) x_j \]

The same

On-line gradient update:

\[ w_0 \leftarrow w_0 + \alpha (y - f(\mathbf{x})) \]

\[ w_j \leftarrow w_j + \alpha (y - f(\mathbf{x})) x_j \]
Limitations of basic linear units

**Linear regression**

$$f(x) = w_0 + \sum_{j=1}^{d} w_j x_j$$

Function linear in inputs !!

**Logistic regression**

$$f(x) = p(y = 1 | x, w) = g(w_0 + \sum_{j=1}^{d} w_j x_j)$$

Linear decision boundary!!

Regression with the quadratic model.

**Limitation:** linear hyper-plane only
- a non-linear surface can be better
Classification with the linear model.

Logistic regression model defines a linear decision boundary
• Example: 2 classes (blue and red points)

Linear decision boundary
• logistic regression model is not optimal, but not that bad
When logistic regression fails?

- Example in which the logistic regression model fails

Limitations of linear units.

- Logistic regression does not work for **parity functions**
  - no linear decision boundary exists

**Solution:** a model of a non-linear decision boundary
Extensions of simple linear units

- use feature (basis) functions to model nonlinearities

### Learning with extended linear units

Feature (basis) functions model nonlinearities

<table>
<thead>
<tr>
<th>Linear regression</th>
<th>Logistic regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x) = w_0 + \sum_{j=1}^{m} w_j \phi_j(x)$</td>
<td>$f(x) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(x))$</td>
</tr>
</tbody>
</table>

**Important property:**  
- The same problem as learning of the weights for linear units, the input has changed– but the weights are linear in the new input  
**Problem:** too many weights to learn
Multi-layered neural networks

- Alternative way to introduce nonlinearities to regression/classification models
- **Idea:** Cascade several simple neural models with logistic units. Much like neuron connections.

**Multilayer neural network**

Also called a **multilayer perceptron (MLP)**

Cascades multiple logistic regression units

**Example:** (2 layer) classifier with non-linear decision boundaries
Multilayer neural network

- Models **non-linearities through logistic regression units**
- Can be applied to both **regression and binary classification problems**

$$\sum_{i=1}^{d} x_i w_i^l = z_i^l(1)$$

Hidden layer

Output layer

**regression**

$$f(x) = f(x, w)$$

**classification**

$$f(x) = p(y = 1 \mid x, w)$$

CS 2750 Machine Learning
Learning with MLP

- How to learn the parameters of the neural network?
- **Gradient descent algorithm**
  - Weight updates based on the error: $J(D, w)$

$$w \leftarrow w - \alpha \nabla_w J(D, w)$$

- We need to **compute gradients for weights in all units**
- **Can be computed in one backward sweep through the net !!!**

![Diagram of neural network](image)

- The process is called **back-propagation**

---

**Backpropagation**

<table>
<thead>
<tr>
<th>(k-1)-th level</th>
<th>k-th level</th>
<th>(k+1)-th level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i(k-1)$</td>
<td>$\sum z_i(k)$</td>
<td>$x_i(k)$</td>
</tr>
<tr>
<td>$w_{i,j}(k)$</td>
<td>$z_i(k)$</td>
<td>$w_{i,j}(k+1)$</td>
</tr>
<tr>
<td></td>
<td>$\sum$</td>
<td>$z_i(k+1)$</td>
</tr>
<tr>
<td></td>
<td>$x_i(k+1)$</td>
<td></td>
</tr>
</tbody>
</table>

$x_i(k)$ - output of the unit $i$ on level $k$

$z_i(k)$ - input to the sigmoid function on level $k$

$w_{i,j}(k)$ - weight between units $j$ and $i$ on levels (k-1) and k

$z_i(k) = w_{i,o}(k) + \sum_j w_{i,j}(k)x_j(k-1)$

$x_i(k) = g(z_i(k))$
Backpropagation

**Update weight** \( w_{i,j}(k) \) using a data point \( D = \{ \textbf{x}, y \} \)

\[
w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D, w)
\]

Let \( \delta(k) = \frac{\partial}{\partial z_i(k)} J(D, w) \)

Then:

\[
\frac{\partial}{\partial w_{i,j}(k)} J(D, w) = \frac{\partial J(D, w)}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k - 1)
\]

S.t. \( \delta_i(k) \) is computed from \( x_i(k) \) and the next layer \( \delta_i(k+1) \)

\[
\delta_i(k) = \left[ \sum_j \delta_j(k+1) w_{i,j}(k+1) \right] x_i(k)(1 - x_i(k))
\]

**Last unit** (is the same as for the regular linear units):

\[
\delta_i(K) = -\sum_u^n (y_u - f(x_u, w))
\]

It is the same for the classification with the log-likelihood measure of fit and linear regression with least-squares error!!!
Learning with MLP

- **Online gradient descent algorithm**
  
  Weight update:

  \[
  w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J_{\text{online}}(D_u, w)
  \]

  \[
  \frac{\partial}{\partial w_{i,j}(k)} J_{\text{online}}(D_u, w) = \frac{\partial J_{\text{online}}(D_u, w)}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_j(k) x_j(k-1)
  \]

  \[
  w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_j(k) x_j(k-1)
  \]

  \(x_j(k-1)\) - j-th output of the (k-1) layer

  \(\delta_j(k)\) - derivative computed via backpropagation

  \(\alpha\) - a learning rate

---

Online gradient descent algorithm for MLP

**Online-gradient-descent** \((D, \text{number of iterations})\)

**Initialize** all weights \(w_{i,j}(k)\)

**for** \(i=1:1: \text{number of iterations}\)

**do**

**select** a data point \(D_u=<x,y>\) from \(D\)

**set** learning rate \(\alpha\)

**compute** outputs \(x_j(k)\) for each unit

**compute** derivatives \(\delta_j(k)\) via backpropagation

**update** all weights (in parallel)

\[
 w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_j(k) x_j(k-1)
\]

**end for**

**return** weights \(w\)
Xor Example.

• linear decision boundary does not exist
Xor example.
Neural network with 2 hidden units

Xor example.
Neural network with 10 hidden units
MLP in practice

- **Optical character recognition** – digits 20x20
  - Automatic sorting of mails
  - 5 layer network with multiple output functions

<table>
<thead>
<tr>
<th>Layer</th>
<th>Neurons</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>3000</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>1200</td>
</tr>
<tr>
<td>3</td>
<td>1200</td>
<td>50000</td>
</tr>
<tr>
<td>2</td>
<td>784</td>
<td>3136</td>
</tr>
<tr>
<td>1</td>
<td>3136</td>
<td>78400</td>
</tr>
</tbody>
</table>

10 outputs (0,1,…,9)

20x20 = 400 inputs