Machine Learning for BCI
The whirlwind tour

CSE599e: Brain-Computer Interfaces

Liberally inspired by

http://autonlab.org/tutorials
(Andrew W. Moore’s slides on all things data-mining)
Outline

- Supervised Learning: Regression
  - Linear, polynomial.
  - RBFs, perceptrons, multilayer networks.
- Supervised Learning: Classification
  - Linear classifiers, max margin, kernels.
  - NN-based classifiers.
- Cross-validation
  - Model selection, preventing overfitting.

Why?

- Dominant paradigms in BCI literature
  - Correlate instantaneous hand position with neural firings ⇒ regression. (Invasive BCIs)
  - Guess whether you’re thinking tomatoes or oranges ⇒ classification. (EEG BCIs)
- In each case, we have example data.
  - Supervised learning.
Linear Regression

Assumption: Output is a linear function of input, i.e.,
\[ y_i = wx_i + \text{noise}_i \]
where noise is independent, gaussian, unknown fixed variance.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.1</td>
</tr>
<tr>
<td>2</td>
<td>6.4</td>
</tr>
<tr>
<td>3.1</td>
<td>8.7</td>
</tr>
<tr>
<td>0.7</td>
<td>2</td>
</tr>
</tbody>
</table>

Thus, \( y_i \) are drawn from \( N(wx_i, \sigma^2) \).
Likelihood of data \((y_i, x_i)\) for a given \(w\),
\[ \Pi_i p(y_i \mid w, x_i) \quad i.e., \]
\[ \Pi_i \exp(-0.5(y_i - wx_i)^2/\sigma^2) \]
Maximize the likelihood of data given \(w\).
i.e., maximize: \[ \Sigma_i -0.5(y_i-wx_i)^2/\sigma^2 \]
i.e., minimize: \[ \Sigma_i (y_i -wx_i)^2 \]

Easy to show that \[ w = \Sigma x_i y_i / \Sigma (x_i)^2 \]
Multivariate regression

Suppose inputs are vectors. We assume:
\[ y = w^T x + \text{noise}. \]
We can write the data points as:
\[
X = \begin{bmatrix}
    x_{11} & x_{12} & \ldots \\
    \vdots \\
    x_{i1} & x_{i2} & \ldots \\
    \vdots
\end{bmatrix}, \quad
Y = \begin{bmatrix}
    y_1 \\
    \vdots \\
    y_i \\
    \vdots
\end{bmatrix}
\]

Then, maximum likelihood \( w \) is
\[ w = (X^T X)^{-1} (X^T Y) \]

Linear regression: constants

What if data does not go through origin?

Simple hack for adding constant term:
\[ y = wx + c \]

Any guesses?
Linear Regression: constants

- Add a dummy input fixed at 1.

\[
\begin{array}{|c|c|}
\hline
x & y \\
\hline
1 & 8.1 \\
2 & 11.4 \\
3.1 & 13.7 \\
0.7 & 7 \\
\hline
\end{array}
\quad \longrightarrow \quad
\begin{array}{|c|c|c|}
\hline
z_1 & z_2 & y \\
\hline
1 & 1 & 8.1 \\
1 & 2 & 11.4 \\
1 & 3.1 & 13.7 \\
1 & 0.7 & 7 \\
\hline
\end{array}
\]

Learn the regression function \( y = w^T z + \text{noise} \).

Regression with Polynomials

- Quadratic, polynomial functions:
  - Same trick: replace input by extended-input

\[
\begin{array}{|c|c|c|}
\hline
x_1 & x_2 & y \\
\hline
... & ... & ... \\
x_{i1} & x_{i2} & y_i \\
... & ... & ... \\
\hline
\end{array}
\quad \longrightarrow \quad
\begin{array}{|c|c|c|c|c|}
\hline
z_1 & z_2 & z_3 & ... & y \\
\hline
... & ... & ... & ... & ... \\
x_{i1} & x_{i2} & (x_{i1})^2 & x_{i1}x_{i2} & y_i \\
... & ... & ... & ... & ... \\
\hline
\end{array}
\]

- Again, learn the model \( y = w^T z + \text{noise} \).
Regression with Polynomials

- Quadratic, polynomial functions:
  - Same trick: replace input by extended-input

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$x_{i1}$</td>
<td>$x_{i2}$</td>
<td>$y_i$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Example: Quadratic Regression

Radial Basis Functions

- Create features that are some function of the entire input vector, e.g.,
  $z_i = \text{KernelFunction}(|x_i - c_i|/\gamma_i)$
- $c_i$ s and $\gamma_i$ s are constant (will revisit).
- Idea: the kernel functions have reasonable overlap, and cover the interesting regions of input space.
RBFs: an example

Function: \( y = 2z_1 + 3.8z_2 + 2.3z_3 \)

Data Point \( y_i \)

Basis Functions \( z_1, z_2, z_3 \)

With \( c_i, y_i \) known, this is standard linear regression.

How to select \( c_i \) and \( y_i \)?

Answer: Gradient Descent.

Perceptrons

Another view of linear regression:

Input \( x \)

Output \( O \)

Learning rule: \( w_k \leftarrow w_k - \eta (O - y) x_{ik} \)

(Where \( \eta \) is the learning rate)

Intuition: Use error to appropriately correct \( w \).

Guarantee: Will converge to ML estimate of \( w \).
Perceptrons: Learning rule(s)

- Essentially performing gradient descent:
  \[ w_k \leftarrow w_k - \eta \frac{\delta \text{error}}{\delta w_k} \]
- Possibly useful for incremental update.
- Idea generalizes to larger networks.
- Useful analogy to brain’s learning mechanisms.

Sigmoid Perceptrons

- Add a sigmoid to the output:
  \[
  \text{Out}(x) = g(w^T x)
  \]

Suitable for classification
- Output is (almost) binary.
- Gradient descent guaranteed to converge, even with sigmoid.
- Performs well in practice.
Multilayer Networks

Perceptron can only express limited class of functions:
\[ \text{out}(x) = g(w^T x) \]
Solution: Chain individual nodes together:
\[ \text{Out}(x) = g(\sum_j w_j g(\sum_k w_{jk} x_k)) \]
Learning is by adaptation of gradient descent—back-propagate errors down the network.

Linear classifiers

Data \( x_i \) are points in \( n \)-dimensional space, with labels +1/-1
Choose a plane \( (w, b) \) separating them.

Then, for any point \( x \), label(\( x \)) = sign(\( w^T x + b \))
Note: can use magnitude of output too, as a measure of plus-ousity.
Linear classifiers: LDA

Assume each class is a **gaussian cloud**
\[ N(\mu_k, \Sigma_k), \quad k = 1,2 \]

Define \( w \) as follows:
\[ w = \Sigma^{-1}(\mu_2 - \mu_1) \]

Easy to compute.
Works if data is sufficiently low-dimensional.

Support Vector Machines

Choose “thickest hyperplane” for \( w \).

Intuitively seems like a good pick, avoids outlier problems.

Can be shown: margin = \( \frac{2}{w^Tw} \)

Maximize margin subject to the following:
\[ w^T x_i + b > +1 \quad \text{for class 1 points,} \]
\[ w^T x_i + b < -1 \quad \text{for class 2 points.} \]
SVMs: Soft Margin

What about outliers? 
Allow errors $\xi_i$

How to control errors? 
Trade off margin with errors.

Minimize $w^T w + C \sum_i \xi_i$ subject to:
$w^T x_i + b + \xi_i > +1$ for class 1 points,
$w^T x_i + b - \xi_i < -1$ for class 2 points,
and $\xi_i > 0$

How to choose $C$?

SVMs: Using Kernels

- Remember trick used for polynomial regression?
  - Replace inputs with functions of inputs:
    $x \leftarrow \Phi(x)$
- Compute a large-margin linear classifier in the “higher-dimensional space” of $\Phi$.
- **Kernel trick**: we only ever need to look at scalar products of points: $\Phi(x_1) \cdot \Phi(x_2)$
- **Kernel trick**: define a kernel function $k(x, y)$, e.g., $\Phi(x) \cdot \Phi(y) = k(x, y) = \exp(-(x-y)^2/2\sigma^2)$
- Can use (almost) arbitrary kfs, where the equivalent $\Phi$ space cannot be represented.
**Multiclass Classifiers**

- Guesses?
- Train $M$ classifiers (for $M$-class problem)
  - class1 vs not-class1, and so on.
  - combine classifier outputs.
- Train $M(M-1)/2$ classifiers:
  - class1 vs class2 for every pair.
  - combine classifier outputs.

---

**Cross-validation**

Question: Which line? The squiggly *overfits* the data.
Answer: *Hold out* some of the points, evaluate performance on the held-out points. (This is what we want it to do in practice)
What if points are (un)lucky?
Cross-validation

- Rinse-repeat-cycle: for each choice of holdout set, do:
  - Do regression on remaining points.
  - Evaluate error on held-out points.
  - Add to total error score.
- Choose model with least generalization error.
- Leave-one-out: each point is used as a holdout set.
- K-fold: Split data into k blocks, each block is used as holdout set.

CV for classification

- Choose model with least generalization error.
  - Can also use to choose parameter of the model (e.g., remember the cost tradeoffs in SVM classifier?)
  - Try a range of values (usually exponentially increasing).
  - Pick parameter value with least CV error.
Fin

References:
http://autonlab.org/tutorials
(Andrew W. Moore)