Generalization and Capacity Control

Léon Bottou

COS 424 – 3/30/2010
Introduction

Generalization
– Optimize system on a training set and expect it to work in future situation.
– Why does it work? When?
– Can we do it better?

Summary
– Generalization: why and when?
– Structural risk minimization.
– Learning algorithms in little pieces.
Generalization is not obvious

Continue the sequence
– 1, 3, 5, 7, ?

Learning by heart versus learning the concept
Generalization is not obvious

Continue the sequence
– 1, 3, 5, 7, ?

Learning by heart versus learning the concept
– But which concept it the right one?

A couple answers
– Odd numbers: 1, 3, 5, 7, 9, 11, 13, 15, 17, . . .
– Prime numbers: 1, 3, 5, 7, 11, 13, 17, 19, 23, . . .
– Numbers palindromic in base two: 1, 3, 5, 6, 9, 15, 17 . . .
– Integers such that $10^n + 19$ is prime: 1, 3, 5, 7, 10, 11, 17, 59, . . .
– Sloane’s encyclopedia of numerical sequences
  lists 598 well known sequences that start like that.
I. Why can we generalize?
The transductive paradigm

For simplicity, we consider only a binary classification problem.

– Given a dataset of $2l$ examples (pattern + class).
– Split examples in two sets $L$ and $T$ of size $l$.
– Training on the learning set $L$ returns a classifier $f$.
– Measure learning error $\mu_L$.
– Measure testing error $\mu_T$.

There are $N_{\text{splits}} = \binom{2l}{l} = \frac{2l!}{l! l!}$ possible splits.

How many splits yield $\mu_T > \mu_L + \epsilon$?

Let’s count them!
Transduction and truth

Ground truth assumption
– “All examples are drawn \textit{independently} from a single \textit{unknown probability distribution}.”
– We cannot test whether this assumption is correct.

Transductive paradigm
– We do not make the ground truth assumption!
– In fact we \textit{do not assume anything about the examples}!
– But we \textit{assume that all splits are equally likely}.
– And we consider that success on the testing set is sufficient evidence.

Remark
– The transduction paradigm \textit{avoids a lot of technical difficulties}.
– Because it describes something more essential...
Error vectors

Classification function $f$.

Examples $(x_1, c_1) \ldots (x_i, c_i) \ldots (x_{2l}, c_{2l})$.

Error vector $m(f) = (0, 0, 0, 1, 0, 0, 1, 0 \ldots , 0, 0, 0)$.

$$m(f) = \begin{cases} 0 & \text{if } x_i \text{ correctly classified by } f \\ 1 & \text{otherwise} \end{cases}$$

Summarizes everything we need to know about $f$.
Enough to compute learning error $\mu_L$ and testing error $\mu_T$. 
When there is a single error vector

Assumption

- Stupid algorithm always returns the same classification function $f$ regardless of the training set $L$.
- Therefore we always get the same error vector $m = m(f)$

Let's count the splits

- Assume vector contains $p$ ones: $p = 2l \mu(m) = l(\mu_T(m) + \mu_L(m))$.
- There are $\binom{k}{p} \binom{l-k}{2l-p}$ splits with $k$ ones in the training set.

$$
\Pr \{\mu_T(m) - \mu_L(m) > \epsilon\} = \Pr \left\{ \frac{p - k}{l} - \frac{k}{l} > \epsilon \right\} = \frac{1}{N_{\text{splits}}} \sum_{p-2k > l \epsilon} \binom{k}{p} \binom{l-k}{2l-p}
$$

- This is called the hypergeometric tail.
Bounds and approximations

We can use a computer and tabulate everything
– Let’s define $\epsilon(\mu, l, \eta)$ such that

$$\Pr \left\{ \mu_T(m) - \mu_L(m) > \epsilon(\mu(m), l, \eta) \right\} = \eta$$

But some bounds and approximations can be useful
– From (Vapnik, 1982) using Chernoff bounding:

$$\epsilon(\mu, l, \eta) \leq \sqrt{\frac{\log(2/\eta)}{l - 1}}$$

$$\epsilon(\mu, l, \eta) \leq \sqrt{4\mu \frac{\log(2/\eta)}{l}}$$

– Reasonable approximation for $l$ large enough:

$$\epsilon(\mu, l, \eta) \approx \sqrt{4\mu(1 - \mu) \frac{\log(2/\eta)}{l}}$$

– They all go to zero when $l$ increases.
Decomposition for the general case

\[ \text{Fr} \{ \mu_T - \mu_L > \epsilon \} = \]

\[ \sum_{L,S} \frac{1}{N_{\text{splits}}} \begin{pmatrix}
0 \\
\vdots \\
0 \\
1 \\
0 \\
\vdots \\
0
\end{pmatrix} \times \begin{pmatrix}
\mathbb{1} \{ \mu_T(m_1) - \mu_L(m_1) > \epsilon \} \\
\vdots \\
0 \\
1 \\
\vdots \\
\mathbb{1} \{ \mu_T(m_N) - \mu_L(m_N) > \epsilon \}
\end{pmatrix} \]

- The sum runs over all the possible splits.
- The green vector indicates which error vector is produced by the classifier returned by running learning algorithm on that split.
- The purple vector indicates which error vectors have an error deviation greater than \( \epsilon \).
Gross bound for the general case

Bound the green vector with a vector with all ones! This sometimes called the union bound.

\[
\text{Fr}\{\mu_T - \mu_L > \epsilon(\mu, l, \eta)\} \leq \eta \mathcal{N}(\mathcal{F}, \mathcal{D})
\]

Equivalently

\[
\text{Fr}\left\{\mu_T - \mu_L > \epsilon \left(\mu, l, \frac{\eta}{\mathcal{N}(\mathcal{F}, \mathcal{D})}\right)\right\} \leq \eta
\]

\(\mathcal{N}(\mathcal{F}, \mathcal{D})\) is the important quantity.

- Family \(\mathcal{F}\) contains all classifiers possibly returned by the algorithm.
- \(\mathcal{N}(\mathcal{F}, \mathcal{D})\) counts the distinct error vectors produced by \(f \in \mathcal{F}\).

With probability \(1 - \eta\)

\[
\mu_T - \mu_L \leq \epsilon \left(\mu, l, \frac{\eta}{\mathcal{N}(\mathcal{F}, \mathcal{D})}\right) \approx \sqrt{4\mu(1 - \mu) \frac{\log(2/\eta) + \log \mathcal{N}(\mathcal{F}, \mathcal{D})}{l}}
\]
The failure mode

- Obviously $N(F, D) \leq 2^{2l}$.

What is happening when $N(F, D) = 2^{2l}$?

- All the possible error vectors are represented.
- Split the classifiers from $F$ into groups of classifiers that perform identically on the learning set.
- Each group contains classifiers that produce all possible error patterns on the testing set.
- Classifiers from a same group perform identically on the learning set. Short of additional information, the training algorithm cannot know which ones work well on the testing set.
- No generalization guarantees.
- But we can be lucky... sometimes...
The finite case

Assumption
“The family of classifiers $\mathcal{F}$ is finite.”

Consequences

- $N(\mathcal{F}, \mathcal{D}) \leq \text{Card}(\mathcal{F}) < 2^{2l}$ when $l$ is large enough.

- $\mu_T - \mu_L \lesssim \sqrt{4\mu(1 - \mu) \frac{\log(2/\eta)}{l} + \frac{\log(\text{Card}\mathcal{F})}{l}}$.

- We can generalize.
- We need $l \gg \log \text{Card}(\mathcal{F})$. 
The infinite case

Vapnik-Chervonenkis combinatorial lemma

Let $m_{\mathcal{F}}(l) = \max_{\{x_1, c_1 \ldots x_l, c_l\}} \mathcal{N}(\mathcal{F}, \{x_1, c_1 \ldots x_l, c_l\})$.

- Either $m_{\mathcal{F}}(l) = 2^l$ for all $l$.
- Or $m_{\mathcal{F}}(l) \leq \left(\frac{le}{h}\right)^h$ where $h$ is the last value such that $m_{\mathcal{F}}(h) = 2^h$.

- Quantity $h$ is called the Vapnik-Chervonenkis of the family $\mathcal{F}$.
  It measures the “capacity” of a family of functions.

(Vapnik and Chervonenkis, 1968) (Sauer, 1972)
Some people unfairly call this lemma “Sauer’s lemma”.

Léon Bottou 15/49 COS 424 – 3/30/2010
The infinite case

What is true for $m_{\mathcal{F}}(l)$ is true for $\mathcal{N}(\mathcal{F}, \mathcal{D})$.

- We cannot generalize when $h = \infty$.
- Otherwise $\mu_T \lesssim \mu_L + \sqrt{4\mu(1 - \mu) \left[ \frac{\log(2/\eta)}{l} + \frac{h}{l} \log \frac{2le}{h} \right]}$.
- We can generalize when $h < \infty$.
- We need $l \gg h$. 
VC dimension versus other capacity measures

Results with VC dimension
- Impressive achievements despite gross bounding technique:
  - Infinite VC dimension $\implies$ no generalization guarantees.
  - Finite VC dimension $\implies$ generalization when $l$ is large enough.
- Price of the gross bounding technique:
  - Simple bounds on $\mu_T - \mu_L$ are way too large.

Annealed VC entropy $\log \mathcal{N}(\mathcal{F}, \mathcal{D})$
- This is a quantity that matters more.
- Better estimates of this quantity improves $\mu_T - \mu_L$ bounds.
- Lots of sophisticated theoretical works.
  - data dependent bounds, localized bounds, ...

Union bound
- Maybe the coarsest bound here.
- Little progress improving on that.
Capacity = number of parameters

The VC dimension of the set of linear discriminant functions

\[ f_{w,b}(x) = \mathbb{I}\{w^\top x + b \geq 0\} \quad x, w \in \mathbb{R}^d \quad b \in \mathbb{R} \]

is equal to the number of parameters

\[ h = d + 1. \]
Capacity >> number of parameters

The VC dimension of the set of functions

\[ f_w(x) = \mathbb{I} \{ \sin(wx) \geq 0 \} \quad x, w \in \mathbb{R} \]

is infinite

![Graph of sine function with points indicating the set of functions with positive sine values.](image-url)
Assume the patterns $x_1 \ldots x_{2l}$ are known beforehand. The classes are unknown.

Let $R = \max \|x_i\|$.

We say that a hyperplane

$$w^\top x + b \quad w, x \in \mathbb{R}^d \quad \|w\| = 1$$

separates patterns with margin $\Delta$ if

$$\forall i = 1 \ldots 2l \quad |w^\top x_i + b| \geq \Delta$$

The family of $\Delta$-margin separating hyperplanes has

$$\log \mathcal{N}(\mathcal{F}, \mathcal{D}) \leq h \log \frac{2le}{h} \quad \text{with} \quad h \leq \min \left\{ \frac{R^2}{\Delta^2}, d \right\} + 1$$
Razors

**Occam's razor**
- When reasoning, “*entities must not be multiplied beyond necessity*”.  
- Means that the simplest solution is often the correct one.

**Capacity and Occam's razor**
- What matters is not the complexity of the final classifier,  
  but the capacity of the family of classifiers we consider.  
- Capacity is not the same as the number of parameters.

**Vapnik's razor**
- “*When solving a problem, avoid solving a more complicated problem as an intermediate step.*”  
- The more complicated problem needs a higher capacity $\mathcal{F}$.  
- Therefore one would need more examples.
II. Structural Risk Minimization, etc.
Structural Risk Minimization

Consider an embedded sequence of families of functions

\[ \mathcal{F}_1 \subset \mathcal{F}_2 \subset \mathcal{F}_3 \subset \ldots \]

This is called a structure.
There is a best family in the sequence for each $l$. 
Model Selection

1. Define a capacity control structure.

2. Optimize for each structure member.

3. **Choose one.**
   - *Empirically:*
     - Holdout (looses examples)
     - Leave-one-out (high variance)
     - K-Fold CV (few results)
   - *Theoretically:*
     - Standard VC bounds ($p < 1000$)
     - Advanced VC bounds ($p < 1$)
     - Effective VC bounds (compute intensive)
     - **Automatic** (dream)
What is a “structure”

The structure defines a preorder on the functions.

All other things being equal:
– We’ll prefer a function from $\mathcal{F}_1$ over a function of $\mathcal{F}_2$.
– We’ll prefer a function from $\mathcal{F}_2$ over a function of $\mathcal{F}_3$.
– We’ll prefer a function from $\mathcal{F}_3$ over a function of $\mathcal{F}_4$.
– etc.

Very similar to a Bayesian prior!
Regularizers

Let $C_1 < C_2 < C_3 < C_4 < \ldots$

Define $\mathcal{F}_i = \{ f : \Omega(f) \leq C_i \}$.

The function $\Omega(f)$ expresses preferences. We prefer $f_1$ over $f_2$ when $\Omega(f_1) < \Omega(f_2)$.

Resulting learning algorithm:

$$
\min_f \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)) + \lambda \Omega(f)
$$

- Regularizer $\Omega(f)$ expresses preferences.
- Hyperparameter $\lambda$ define their strength.
- Choosing $\lambda$ amounts to choosing a $\mathcal{F}_i$.
- Must adjust $\lambda$ for each $l$, for instance using cross-validation.
III. Learning algorithms in little pieces
## A. Representation

<table>
<thead>
<tr>
<th>Representation</th>
<th>Parametric vs. Kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linear vs. Non-Linear</td>
</tr>
<tr>
<td>Capacity Control</td>
<td>Explicit via architecture</td>
</tr>
<tr>
<td></td>
<td>Explicit via feature selection</td>
</tr>
<tr>
<td></td>
<td>Explicit via regularizers</td>
</tr>
<tr>
<td></td>
<td>Implicit via optimization</td>
</tr>
<tr>
<td></td>
<td>Implicit via margins</td>
</tr>
<tr>
<td>Operational Considerations</td>
<td>Loss functions</td>
</tr>
<tr>
<td></td>
<td>Online vs. offline</td>
</tr>
<tr>
<td></td>
<td>Budget constraints</td>
</tr>
<tr>
<td>Computational Considerations</td>
<td>Exact algos for <em>small data.</em></td>
</tr>
<tr>
<td></td>
<td><em>Stochastic algos for big data.</em></td>
</tr>
<tr>
<td></td>
<td>Parallel algos.</td>
</tr>
</tbody>
</table>
**Representation - Parametric**

Parametric:

- \( \theta \) is a vector.

Non Parametric:

- \( \theta \) belongs to a bigger space.
Representation - Networks

Example: Convolutional Network

- Very good for *image and signal*.
- Learning algorithms are *delicate* (non-linear, non-convex.)
Representation - Centroids

Examples: KMeans, LVQ

- Fast algorithms for relatively low input dimension.
- Generally more difficult for high input dimension.
**Representation - Mixtures**

Example: Mixture of Gaussians

- Fast algorithms for relatively low input dimension.
- Generally more difficult for high input dimension.

Example: Mixture of Experts, Voting Schemes.

- Divide and conquer.
- Capacity control more difficult.
Representation - Kernels

Suitable for $\theta$ in function space.

- Decision function uses the training examples $x_i$.

\[ \hat{y}(x) = \sum_i \alpha_i K(x_i, x) + b \]

- Kernel function is a dot-product in some large space.

\[ K(x, y) = \langle \Phi(x), \Phi(y) \rangle \]

- Algorithms must only use $K(x, y)$, not $\Phi(x)$.

- Sparsity is desirable.

- See next lecture!
### B. Loss Functions

| Representation                               | Parametric vs. Kernels  
|                                            | Linear vs. Non-Linear |
|                                            |                      |
| Capacity Control                             | Explicit via architecture  
|                                            | Explicit via feature selection |
|                                            | Explicit via regularizers  
|                                            | Implicit via optimization |
|                                            | Implicit via margins |

| Operational Considerations                  | Loss functions  
|                                            | Online vs. offline |
|                                            | Budget constraints |

| Computational Considerations                | Exact algos for *small data*.  
|                                            | *Stochastic algos for* big data. |
|                                            | Parallel algos. |
Pattern Recognition Losses - Ideal

- Example $x$ has class $y = \pm 1$.
- Mistake if $y$ and $\hat{y}(x)$ have different signs.
- Minimize number of mistakes?
Pattern Recognition Losses - Quadratic

\[ L = (y - \hat{y}(x))^2 \]

- Approximate posterior probabilities \( P(y|x) \).
- Convex, easy to optimize.
- Bound problems.
Pattern Recognition Losses - Sigmoid

\[ L = (y - 1.7 \tanh(\hat{y}(x)))^2 \]

- Solves bound problems.
- Non-convex, more difficult to optimize.
- Still approximate posterior probabilities \( P(y|x) \).
Centroid Representation + LVQ loss $\rightarrow$ LVQ algorithm.

$$\hat{y}(x) = \frac{(x - w^+)^2 - (x - w^-)^2}{\delta(x - w^-)^2}$$

$w^-$: closest centroid.

$w^+$: closest centroid w/correct class.
Pattern Recognition Losses - Exp/Log

- Approximate posterior probabilities $P(y|x)$.
- Convex.

- ExpLoss $\leftrightarrow$ Boosting, . . .
- LogLoss $\leftrightarrow$ Maximum likelihood for $P\{Y|X\}$.
Pattern Recognition Losses - Hinge

- Does not approximate probabilities.
- Convex.

\[ y \hat{y}(x) \]

- **PerceptronLoss** ↔ Perceptron,…
- **HingeLoss** ↔ SVM,…
Pattern Recognition Losses - Ramps
Minimize quantization error:

\[ L = \min_k (x - w_k)^2 \]

Example: k-Means.
**Density Losses - KL, Hellinger**

Optimize:

\[ L = - \log(\hat{y}(x)) \]

Normalization:

\[ \int \hat{y}(x) \, dx = 1 \quad \rightarrow \quad P(x) \]
\[ \sum_k \hat{y}_k(x) = 1 \quad \rightarrow \quad P(y|x) \]

Most statistical systems.

---

Compare histograms:

\[ L = \sum_k \left( \sqrt{\hat{y}_k(x)} - \sqrt{y_k} \right)^2 \]

Novelty detection.
## C. Capacity Control Strategies

<table>
<thead>
<tr>
<th>Representation</th>
<th>Parametric vs. Kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Linear vs. Non-Linear</td>
</tr>
<tr>
<td>Capacity Control</td>
<td>Explicit via architecture</td>
</tr>
<tr>
<td></td>
<td>Explicit via feature selection</td>
</tr>
<tr>
<td></td>
<td>Explicit via regularizers</td>
</tr>
<tr>
<td></td>
<td>Implicit via optimization</td>
</tr>
<tr>
<td></td>
<td>Implicit via margins</td>
</tr>
<tr>
<td>Operational Considerations</td>
<td>Loss functions</td>
</tr>
<tr>
<td></td>
<td>Online vs. offline</td>
</tr>
<tr>
<td></td>
<td>Budget constraints</td>
</tr>
<tr>
<td>Computational Considerations</td>
<td>Exact algos for <em>small data</em>.</td>
</tr>
<tr>
<td></td>
<td><em>Stochastic algos for</em> big data.</td>
</tr>
<tr>
<td></td>
<td>Parallel algos.</td>
</tr>
</tbody>
</table>
Capacity Control - Explicit

via Architecture
Architecture changes capacity.
Model selection.
Search algorithm.

via Feature Selection
Feature selection.
Search algorithms.
**Capacity Control - Explicit**

via Preprocessing
Handcrafted features.
Dimensionality reduction.
Smoothing.

\[
\min \left\{ \mathbb{E}_z L(z, w) + \lambda \Omega(w) \right\}
\]

via Regularization
Hyperparameter \( \lambda \).

<table>
<thead>
<tr>
<th>( \Omega )</th>
<th>Ridge, ( \ldots )</th>
<th>Lasso, ( \ldots )</th>
<th>Smoothness, ( \ldots )</th>
<th>Tangent prop</th>
</tr>
</thead>
<tbody>
<tr>
<td>( |w|_2^2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( |w| )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( |\partial_x \hat{y}(x)|_2^2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( |T_\alpha w|_2^2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Capacity Control - Implicit

Example:

- Non-Linear Neural Network.
- $10^5$ to $10^6$ parameters.
- Poor optimizer (stochastic gradient)

This controls capacity via:

- Initial parameters $w_0$.
- Learning rate limits $\|w - w_0\|^2$.
- Algorithm inefficiencies $\|\partial_x \hat{y}(x)\|^2$.
- Early stopping.

The capacity control levers are mixed with the delicate optimization settings.
Bayesian priors also express a preorder on the functions $f$.
– but also define a numerical strength associated with preferences.

Bayes framework does not suggest a $\lambda$ hyperparameter.
– But the prior numerical strength can be more or less peaky.
– Often the prior peakiness is controlled by an hyperparameter
  that itselfs obeys its own prior distribution, etc.

Bayesian averaging has no equivalent here!