Introduction to Machine Learning

Model Validation and Selection

Dr. Ilija Bogunovic
Learning and Adaptive Systems (las.ethz.ch)
Recap: Achieving generalization

- Fundamental assumption: Our data set is generated independently and identically distributed (iid) from some unknown distribution $P$

  $$(x_i, y_i) \sim P(X, Y)$$

- Our goal is to minimize the expected error (true risk) under $P$

  $$R(w) = \int P(x, y)(y - w^T x)^2 dx dy$$

  $$= \mathbb{E}_{x, y} [(y - w^T x)^2]$$
Recap: Evaluating predictive performance

- Training error (empirical risk) systematically underestimates true risk

$$\mathbb{E}_D \left[ \hat{R}_D(\hat{w}_D) \right] < \mathbb{E}_D \left[ R(\hat{w}_D) \right]$$
Recap: More realistic evaluation?

- Want to avoid underestimating the prediction error
- **Idea:** Use separate test set from the same distribution $P$ 
- Obtain training and test data $D_{train}$ and $D_{test}$
- Optimize $w$ on training set

$$\hat{w} = \arg\min_w \hat{R}_{train}(w)$$

- Evaluate on test set

$$\hat{R}_{test}(\hat{w}) = \frac{1}{|D_{test}|} \sum_{(x,y) \in D_{test}} (y - \hat{w}^T x)^2$$

- Then:

$$\mathbb{E}_{D_{train}, D_{test}} \left[ \hat{R}_{D_{test}}(\hat{w}_{D_{train}}) \right] = \mathbb{E}_{D_{train}} \left[ R(\hat{w}_{D_{train}}) \right]$$
$D_{\text{train}} = D, \quad D_{\text{test}} = V, \quad D, V \sim \mathcal{P}$

$E_{0, V} [\hat{R}_V(\hat{w}_0)] \equiv E_0 [E_V [\hat{R}_V(\hat{w}_0)]]$ (ind. of $D, V$)

$= E_0 \left[ E_V \left[ \frac{1}{|V|} \sum_{i=1}^{|V|} (y_i - \hat{w}_0^T x_i)^2 \right] \right]$ (Def. of $\hat{R}_V(\cdot)$)

$= E_0 \left[ \frac{1}{|V|} \sum_{i=1}^{|V|} E_{x_i, y_i} (y_i - \hat{w}_0 x_i)^2 \right]$

$= E_0 \left[ R(\hat{w}_0) \right]$ \hspace{2cm} $R(\hat{w}_0), (x_i, y_i) \perp \independent$

$\square$
Recap: Evaluating predictive performance

- Training error (empirical risk) systematically underestimates true risk

\[ \mathbb{E}_D \left[ \hat{R}_D(\hat{w}_D) \right] < \mathbb{E}_D \left[ R(\hat{w}_D) \right] \]

- Using an independent test set avoids this bias

\[ \mathbb{E}_{D_{\text{train}}, D_{\text{test}}} \left[ \hat{R}_{D_{\text{test}}}(\hat{w}_{D_{\text{train}}}) \right] = \mathbb{E}_{D_{\text{train}}} \left[ R(\hat{w}_{D_{\text{train}}}) \right] \]
First attempt: Evaluation for model selection

- Obtain training and test data $D_{\text{train}}$ and $D_{\text{test}}$
- Fit each candidate model (e.g., degree $m$ of polynomial)

$$ \hat{w}_m = \arg\min_{w: \text{degree}(w) \leq m} \hat{R}_{\text{train}}(w) $$

- Pick one that does best on test set:

$$ \hat{m} = \arg\min_{m} \hat{R}_{\text{test}}(\hat{w}_m) $$

- Do you see a problem?
Overfitting to test set

- Test error is itself random! Variance usually increases for more complex models.
- Optimizing for *single* test set creates bias.
Solution: Pick multiple test sets!

- **Key idea:** Instead of using a single test set, use multiple test sets and average to decrease variance!

- **Dilemma:**
  Any data I use for testing I can’t use for training

- Using multiple independent test sets is expensive and wasteful
Evaluation for model selection

For each candidate model $m$ (e.g., polynomial degree) repeat the following procedure for $i = 1:k$

- Split the same data set into training and validation set

$$D = D^{(i)}_{\text{train}} \oplus D^{(i)}_{\text{val}}$$

- Train model

$$\hat{w}_{i|m} = \arg \min_w \hat{R}^{(i)}_{\text{train}}(w)$$

- Estimate error

$$\hat{R}^{(i)}_m = \hat{R}^{(i)}_{\text{val}}(\hat{w}_{i|m})$$

- Select model:

$$\hat{m} = \arg \min_m \frac{1}{k} \sum_{i=1}^{k} \hat{R}^{(i)}_m$$
How should we do the splitting?

- Randomly (Monte Carlo cross-validation)
  - Pick training set of given size uniformly at random
  - Validate on remaining points
  - Estimate prediction error by averaging the validation error over multiple random trials

- k-fold cross-validation (default choice)
  - Partition the data into $k$ „folds“
  - Train on $(k-1)$ folds, evaluating on remaining fold
  - Estimate prediction error by averaging the validation error obtained while varying the validation fold
k-fold cross-validation

\[ D_1 \quad D_2 \quad \ldots \quad D_i \quad \ldots \quad D_k \]
Accuracy of cross-validation

- Cross-validation error estimate is very nearly unbiased for large enough $k$

- *Show demo*
Cross-validation

How large should we pick $k$?

Too small

- Risk of overfitting to test set
- Using too little data for training
- Risk of underfitting to training set

Too large

- In general, better performance! $k=n$ is perfectly fine (called leave-one-out cross-validation, LOOCV)
- Higher computational complexity

In practice, $k=5$ or $k=10$ is often used and works well
Best practice for evaluating supervised learning

- Split data set into training and test set
- Never look at test set when fitting the model. For example, use $k$-fold cross-validation on training set
- Report final accuracy on test set (but never optimize on test set)!

**Caveat:** This only works if the data is i.i.d.

- Be careful, for example, if there are temporal trends or other dependencies
## Supervised learning summary so far

<table>
<thead>
<tr>
<th>Representation/features</th>
<th>Linear hypotheses, nonlinear hypotheses through feature transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model/objective:</td>
<td>Loss-function</td>
</tr>
<tr>
<td></td>
<td>Squared loss, $l_p$-loss</td>
</tr>
<tr>
<td>Method:</td>
<td>Exact solution, Gradient Descent</td>
</tr>
<tr>
<td>Evaluation metric:</td>
<td>Mean squared error</td>
</tr>
<tr>
<td>Model selection:</td>
<td>K-fold Cross-Validation, Monte Carlo CV</td>
</tr>
</tbody>
</table>
Model selection more generally

- For polynomial regression, model complexity is naturally controlled by the degree.
- In general, there may not be an ordering of the features that aligns with complexity.
  - E.g., how should we order words in the bag-of-words model?
  - Collection of nonlinear feature transformations:
    \[ x \mapsto \log(x + c) \]
    \[ x \mapsto x^\alpha \]
    \[ x \mapsto \sin(ax + b) \]

- Now model complexity is no longer naturally „ordered“.
Demo: Overfitting $\rightarrow$ Large Weights
Regularization

- If we only seek to minimize our loss (optimize data fit) can get very complex models (large weights)

- Solution?

  - Regularization!
    Encourage small weights via penalty functions (regularizers)
Ridge regression

- Regularized optimization problem:
  \[ \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \| \mathbf{w} \|_2^2 \]
  \[ \lambda > 0 \]

- Can optimize using gradient descent, or still find analytical solution:
  \[ \hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \]

- Note that now the scale of \( \mathbf{x} \) matters!
Renormalizing data: Standardization

Ensure that each feature has zero mean and unit variance

\[ \tilde{x}_{i,j} = \frac{(x_{i,j} - \hat{\mu}_j)}{\hat{\sigma}_j} \]

Hereby \( x_{i,j} \) is the value of the j-th feature of the i-th data point

\[
\hat{\mu}_j = \frac{1}{n} \sum_{i=1}^{n} x_{i,j} \\
\hat{\sigma}^2_j = \frac{1}{n} \sum_{i=1}^{n} (x_{i,j} - \hat{\mu}_j)^2
\]
Gradient descent for ridge regression

\[ \nabla_w \left( \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|^2 \right) = \nabla_w \hat{R}(w) + \lambda \nabla_w \|w\|^2 \\
\hat{R}(w) \\
\|w\|^2 \\
\left( \|w\|^2 = w^T w \right) \]

\[ w_{t+1} \leftarrow w_t - \eta_t \left( \nabla_w \hat{R}(w_t) + 2\lambda w_t \right) \]

\[ = (1 - 2\lambda \eta_t) w_t - \eta_t \nabla_w \hat{R}(w_t) \]
Demo: Regularization
How to choose regularization parameter?

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_2^2$$

- Cross-validation!
- Typically pick $\lambda$ logarithmically spaced:
  $$\left\{ 10^{-6}, 10^{-5}, \ldots, 10^5, 10^6 \right\}$$
Regularization path

Ridge coefficients as a function of the regularization

weights

Lambda

$\text{R}^2$
Outlook: Fundamental tradeoff in ML

- Need to trade loss (goodness of fit) and simplicity
- A lot of supervised learning problems can be written in this way:

\[
\min_{\mathbf{w}} \hat{R}(\mathbf{w}) + \lambda C(\mathbf{w})
\]

- Can control complexity by varying regularizer parameter \( \lambda \)
- Many other types of regularizers exist and are very useful (more later in this class)
<table>
<thead>
<tr>
<th>Supervised learning summary so far</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation/features</td>
</tr>
<tr>
<td>Model/objective:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Method:</td>
</tr>
<tr>
<td>Evaluation metric:</td>
</tr>
<tr>
<td>Model selection:</td>
</tr>
</tbody>
</table>
What you need to know

- **Linear regression** as model and optimization problem
  - How do you solve it?
  - Closed form vs gradient descent
  - Can represent non-linear functions using basis functions

- **Model validation**
  - Resampling; Cross-validation

- **Model selection** for regression
  - Comparing different models via cross-validation

- **Regularization**
  - Adding penalty function to control magnitude of weights
  - Choose regularization parameter via cross-validation