Machine Learning for Economists: Part 1 – Learning

Michal Andrle
International Monetary Fund

Washington, D.C., October, 2018
Disclaimer #1:

The views expressed herein are those of the authors and should not be attributed to the International Monetary Fund, its Executive Board, or its management.
Learning ≠ Fitting

**Learning**: Choose a model with sufficient *representational capacity* that will perform well out-of-sample.

For given data, increasing model’s capacity increases **over-fitting**.

**Overfitting:**

- Fits available data, does not **generalize well** to new data from the population (nature’s) distribution
- Small with **training sample** error, large **test sample** error
- Occurs when model complexity too large for **available** data sample (low ‘degrees of freedom’)

Avoiding overfitting is one of top issues in all types of machine learning and deep learning.
“With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.”

John von Neumann [quoted by Enrico Fermi]
Capacity Selection: Extreme Curve Fitting

For \( \{x_1, \ldots, x_n\} \) and \( \{y_1, \ldots, y_n\} \), choose a model

\[
y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_M x_i^M = f(x, \beta)
\]

to minimize

\[
err = \sum_{i}^{N} \{y_i - f(x, \beta)^2\}.
\]

However:

Any \( n \) points can be fit exactly by \( n - 1 \) degree polynomial!

In-sample Fit: Perfect

Out-of-Sample Fit: Disaster!

Note: These are not orthogonal polynomials.
Capacity Selection: Over-fitting

M=1

M=2

M=3

M=9
Capacity Selection: Over-fitting

M=1

M=2

M=3

M=9
Capacity Selection: Over-fitting

![Graphs showing capacity selection](image-url)
Capacity Selection: Over-fitting

- $M=1$
- $M=2$
- $M=3$
- $M=9$
Capacity Selection: Getting More Observations...
Model Order Selection ("Structural Stabilization")

Estimate using training sample, o, and choose the best model on a test sample, +
Capacity Selection: Over-fitting

Very unsettling...

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>M = 1</th>
<th>M = 4</th>
<th>M = 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta_0$</td>
<td>0.746</td>
<td>-0.112</td>
<td>0.00048</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>-1.57</td>
<td>11.139</td>
<td>6.276</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>-32.621</td>
<td>0.157</td>
<td></td>
</tr>
<tr>
<td>$\beta_3$</td>
<td>21.635</td>
<td>-43.086</td>
<td></td>
</tr>
<tr>
<td>$\beta_4$</td>
<td></td>
<td>10.871</td>
<td></td>
</tr>
<tr>
<td>$\beta_5$</td>
<td></td>
<td>39.874</td>
<td></td>
</tr>
<tr>
<td>$\beta_6$</td>
<td></td>
<td>102.809</td>
<td></td>
</tr>
<tr>
<td>$\beta_7$</td>
<td></td>
<td>-239.546</td>
<td></td>
</tr>
<tr>
<td>$\beta_8$</td>
<td></td>
<td>157.725</td>
<td></td>
</tr>
<tr>
<td>$\beta_9$</td>
<td></td>
<td>-35.089</td>
<td></td>
</tr>
</tbody>
</table>

+ In-sample fit is great
  - Really bad generalization
  - Unstable coefficients
  - Predictors highly correlated

What’s happening?
Try to **regularize** the problem!
**Capacity Selection: Over-fitting**

Very unsettling...

<table>
<thead>
<tr>
<th>Coefficients</th>
<th>$M = 1$</th>
<th>$M = 4$</th>
<th>$M = 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta_0$</td>
<td>0.746</td>
<td>-0.112</td>
<td>0.00048</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>-1.57</td>
<td>11.139</td>
<td>6.276</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>-32.621</td>
<td>0.157</td>
<td>-43.086</td>
</tr>
<tr>
<td>$\beta_3$</td>
<td>21.635</td>
<td>10.871</td>
<td>39.874</td>
</tr>
<tr>
<td>$\beta_4$</td>
<td></td>
<td>102.809</td>
<td>-239.546</td>
</tr>
<tr>
<td>$\beta_5$</td>
<td></td>
<td></td>
<td>157.725</td>
</tr>
<tr>
<td>$\beta_6$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\beta_7$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\beta_8$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\beta_9$</td>
<td></td>
<td></td>
<td>-35.089</td>
</tr>
</tbody>
</table>

- In-sample fit is great
  - Really bad generalization
  - Unstable coefficients
  - Predictors highly correlated

What's the solution? Try to **regularize** the problem!
Regularization

“Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

Goodfellow et al. 2017 (pp. 117, Deep Learning)

Regularization embeds a set of priors for solutions into the learning process.

As $N \to \infty$ the model puts less and less weight on the prior restrictions, adapting to data, and increasing effective capacity
Regularization: Introducing prior constraints to solve ill-conditioned problems.

Let’s try to penalize the size of the regression coefficients to prevent overfitting...

New problem to minimize:

\[
err = \sum_{i}^{N} \{y_i - f(x, \beta)\}^2 + \lambda \left\{ \sum_{j=1}^{M} (\beta_j - 0)^2 \right\}.
\]

Hyperparameter \( \lambda \) (tuning parameter) determines the size of the penalty.

\[
\lambda = \begin{cases} 
0 & \text{standard least-squares} \\
(0, \infty) & \text{regularized regression} \\
\infty & \beta_1 = \beta_2 = \cdots = \beta_M = 0
\end{cases}
\]

YES,...this IS very Bayesian, with a prior \( \beta_i \sim N(0, 1/\lambda) \)
Regularization: Choosing Complexity

We can estimate models with general models with polynomial order $M = 9$ and $\lambda = \{1, \ldots, \lambda_{max}\}$ on the training data and evaluate each model on test data.
We can estimate models with general models with $M = 9$ and $\lambda = \{1, \ldots, \lambda_{max}\}$ on the training data and evaluate each model on test data.
Regularized model with order $M = 9$, 11 observations and complexity determined using the test data searching for ‘optimal’ $\lambda$
Regularization: Choosing Complexity

Regularized model with order $M = 9$, 111 observations and complexity determined using the test data searching for ‘optimal’ $\lambda$
Overfitting Classifiers...
The model must strike a balance between bias and variance over the new samples... [from the true, unknown, and hopefully stable DGP]
Apart from various parameter priors, is there anything else? How about estimating many models using boostrapped data and aggregating their forecast? **BAGGING!**

**Bootstrap Aggregating:**
- Common model ensemble method to lower variance
- Useful for unstable estimators, often used with **trees** (Random Forests)

**Our case:**
- Re-sample data with replacement $R$-times, $R = 90$
- Aggregate the model estimates, $\hat{y}_r$ using **median**
Regularization (2b) – Beyond Parameter Priors
Can we try something else? How about **data augmentation**?

**Data augmentation:**
Add modified/corrupted data to the sample to avoid overfitting.

Examples:
- Add rotated images to the sample
- Add images/sounds corrupted by white noise to the sample
- Attempt to address unbalanced datasets for classification (SMOTE, etc.)

**Our case:**
Select 1/3 of the sample, create new data points by adding 
\( \varepsilon_i \sim N(0, \sigma) \) to input \( x_i \), keeping output, \( y_i \) unchanged...
Add **noise** to portion of the data and **augment** the dataset. . .

Note: This is not an accident. Bishop (1998) shows training with noise amounts to L2 penalty regularization.
**Regularization – Summary**

Regularization process is adaptive and can take many forms.

Important examples:

- **Coefficient penalties**, sparsity shrinkage, parameter sharing, …
- Bootstrap aggregating (‘Bagging’) of models
- **Random perturbations** to model structure, model averaging, early stopping
- Artificial feature corruption
  (dropout, data augmenting, adding noise)
- **Theory** – putting a weight on a-priori theory, parameter sharing, …
Regularization – Hyper Parameter ‘Tuning’

Regularization process introduces hyper parameters.

Hyper-parameters cannot be optimized using in-sample fit.

Regularization may WORSEN in-sample fit, to help IMPROVE out-of-sample fit.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Hyperparameter Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>parametric models</td>
<td>weight-decay penalties, $\sum_{i=1}^{p}</td>
</tr>
<tr>
<td>trees, random forests</td>
<td>depth of trees, number of trees, minimum leaf size, ...</td>
</tr>
<tr>
<td>nearest-neighbor</td>
<td>number of neighbors</td>
</tr>
<tr>
<td>neural nets</td>
<td>weight-decay, drop-out, layer architecture</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Regularization – Hyper Parameter ‘Tuning’

How to set the “right values” of tuning parameters?

1. **Setting them a priori**, as priors.

2. **Optimizing them** using different data than used for model training. (validation sample, cross-validation, . . .)

Get used to watching charts like this:
Fixing Overfitting

1. **Use more data**
   Given the model, increase in informative data size lowers overfitting

2. **Structural stabilization**
   Lower the number of free parameters, simplify complexity
   - Inflexible, doesn’t adjust to data
   - Lowering model-hypothesis space may exclude ‘truth’

3. **Regularization**
   Keep using complex model, impose prior restrictions on behavior . . .
   + Keeps large model-hypothesis space, stays flexible
   + Effective representational capacity adapts to information in the data
Bias-Variance Trade-Off

The frequentist view on model complexity is about bias-variance trade-off. 

Treating the given data as a draw from an unknown population, the expected mean-square (population) error is a combination of bias and variance.

1. More complex model may have lower bias but large variance if re-estimated on new draws from the population

2. Less complex model may have higher bias but lower variance, being less sensitive to new draws from the population

In Bayesian setting when one marginalizes over parameters the over-fitting phenomenon is largely absent...
Bias-Variance Trade-Off

Complex Model: Lower Bias, Higher Variance

Simple Model: Higher Bias, Lower Variance
Bias-Variance Trade-Off

Decomposing population expected mean-square error:

expected loss = bias^2 + variance + irreducible noise

To minimize Expected MSE:

Trade lower variance for higher bias, if possible

By Gauss-Markov theorem, least squares have the smallest variance among all the unbiased estimates... If \( \hat{\beta} \) has large variance, \( \hat{y} = \hat{\beta}'x \) has high variance too

...statistical learning trades bias for lower variance, in order for models to generalize well to new data
Bias-Variance Trade-Off

Let’s $y = f(x)$. For a given estimate $\hat{y} = \hat{f}(x)$, we assess population mean squared error

$$E[(y - \hat{f}(x))^2] = E[y^2] + E[\hat{f}^2] - 2E[y\hat{f}]$$

$$= (\text{var}[y] + E[y]^2) + (\text{var}[\hat{f}] + E[\hat{f}]^2) - 2fE[\hat{f}]$$

$$= \text{var}[y] + \text{var}[\hat{f}] + (f - E[\hat{f}])^2$$

$$= \sigma^2 + \text{variance} + \text{bias}$$
Bias-Variance Trade-Off

\[ E[\text{RMSE}] \]

\[ \text{bias} \quad \text{variance} \quad \text{bias+variance} \quad \text{test set (bias + variance + noise)} \]
Say, if using a 0-1 loss, \( L(\hat{y}, y) = I(\hat{y} = y) \), bias and variance combine multiplicatively...

With 0-1 loss, estimation errors that leave you making the right 0/1 decision are not painful. Also, when on the wrong side of the decision boundary with a negative bias, it pays off to *increase* variance.

The choice of **hyperparameters** (tuning parameters) and/or model selection thus should rather focus on estimation of **expected loss**.
The regularization examples were meant to provide intuition...

For **LINEAR regression**, the ridge regression, data augmentation, bagged data-augmentation, or feature dropout all imply an $L_2$ penalty... [if you do all the algebra]

For more complex, nonlinear models, the regularization penalties implied by these techniques start to significantly differ...

Different forms of regularization are suitable for different forms of training, given the problem and the computation tools used

See Wager, Wang, and Liang (Nov 2013)
SUPERVISED LEARNING PROBLEM: SUMMARY
The Elements of Learning

The Problem:
An unknown mapping, $u$, from input space, $\mathcal{X}$, to output space, $\mathcal{Y}$,
$u : \mathcal{X} \rightarrow \mathcal{Y}$, such that $y_j = f(x_j)$.

Hypothesis Set:
A set $\mathcal{R}(f)$ containing candidate mappings (models, hypotheses) mean to
approximate the unknown mapping $u$.

Observed Data:
A finite set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$, or $\mathcal{D} = \{(x_1, x_2), \ldots, (x_k, x_k)\}$.

Algorithm:
An algorithm, $\mathcal{A}$, that uses observed data, $\mathcal{D} \in$, to learn a mapping
$f : \mathcal{X} \rightarrow \mathcal{Y}, f \in \mathcal{R}(f)$.

Error Measure (Distance/Loss/Utility Function)
A metric $\mathcal{E}$ used by the algorithm $\mathcal{A}$ to choose a mapping $g$. 
Machine Learning

\[ \hat{f}(.):= \arg \min_{f \in F_A} \sum_{i=1}^{N} D[y_i, f(x_i); \beta] \quad \text{s.t.} \quad R(f) \leq C \]

- **Classification**: \( y_i \in \{0, 1\} \)
- **Regression**: \( y_i \in \mathcal{R} \)

**Loss Function**

**Parameters**

**Train**

\( \hat{y}_i \) or \( \hat{p}(y_i|x_i) = 1 \)

**Predict**

**Model Complexity**

**(Cross) Validate**

- Loss: \( D[y, f(x); \beta] \)

- Complexity: \( C \)

**Overfitting**

- In sample
- Out of sample
Supervised Learning = Training + Validation

1. Using **training data** optimize for $f(.)$

2. Using **validation set** search hyper-parameters to optimize out-of-sample fit

Choose $\widehat{f}(.)$ such that

$$
\widehat{f}(.) = \arg\min_{f \in F_A} \sum_{i=1}^{N} D[y_i, f(x_i)] \quad \text{s.t.} \quad R(f) \leq c
$$

$$
= \arg\min_{f \in F_A} \sum_{i=1}^{N} D[y_i, f(x_i)] - \lambda\{R(f) - c\}
$$

- $D(.)$ loss function due to difference between $y_i$ and $f(x_i)$
- $\beta$ coefficients to be estimated
- $F_A$ function space with architecture $A$
- $R(f)$ regularizer of the function $f(.)$
- $\lambda$ regularization tightness, “shadow price” . . .
Supervised Learning Problem Defined (2)

Example: **Bayesian [V]AR** (ridge-regression with $K$ lags)

\[
\hat{f}(.) = \arg\min_{f \in \mathcal{F}_A} \sum_{i=1}^{N} D[y_i, f(x_i)] - \lambda \{ R(f) - c \}
\]

$D(.)$  Quadratic loss, $[y_i - f(x_i)]^2$

$\mathcal{F}_A$  Linear model with coefs $\beta$, $K$ lags: $y_v = f(x) = \beta_0 + \beta_1 y_{t-1} + \cdots + \beta_k y_{t-k}$

$R(f)$  Ridge (Normal prior): $\sum_{j=1}^{K} (\beta_{ij} - 0)^2 \times j^{\lambda_1}$

$\lambda$  A-Priori Regularization tightness,

$\lambda_0$ – overall tightness, $\lambda_1$ – higher lag, larger shrinkage to zero

**LEARNING:**

1) For **given** $D$, $\mathcal{F}_A$, $R(f)$ and $\lambda$ use **training data** to find $\beta$.
2) Use **validation data set** to search for $\lambda$, number of lags, etc.
Thank you for your patience...