Statistical Machine Learning
Hilary Term 2020

Pier Francesco Palamara
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
http://www.stats.ox.ac.uk/~palamara/SML20.html

February 11, 2020
Last time: Overfitting, model selection

Fitting the housing price data with high order polynomials

Note that the price would go to zero (or negative) if you buy bigger ones! This is called poor generalization/overfitting.

\[ R(f) = R_N^{\text{emp}}(f) + \text{overfit penalty}. \]

- Cross-validation can be used to estimate \( R(f) \) and select the adequate model complexity.
- Another possible strategy is to try to estimate the overfit penalty (e.g. via regularization).
Building a machine learning model involves trading between its bias and variance. We will see many examples in the next lectures:

- Bias reduction at the expense of a variance increase: building more complex models, e.g. adding nonlinear features and additional parameters, increasing the number of hidden units in neural nets, using decision trees with larger depth, decreasing the regularization parameter.
- Variance reduction at the expense of a bias increase: early stopping, using k-nearest neighbours with larger k, increasing the regularization parameter.
Regularization

- Flexible models for high-dimensional problems require many parameters.
- With many parameters, learners can easily overfit.
- **Regularization**: Limit flexibility of model to prevent overfitting.
- Add term **penalizing large values of parameters** $\theta$.

\[
\min_{\theta} R_N(f_\theta) + \lambda \|\theta\|_\rho = \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda \|\theta\|_\rho
\]

where $\rho \geq 1$, and $\|\theta\|_\rho = (\sum_{j=1}^{p} |\theta_j|^\rho)^{1/\rho}$ is the $L_\rho$ norm of $\theta$ (also of interest when $\rho \in [0, 1)$, but is no longer a norm).

- Also known as **shrinkage** methods—parameters are shrunk towards 0.
- $\lambda$ is a **tuning parameter** (or hyperparameter) and controls the amount of regularization, and resulting complexity of the model.
$L_\rho$ regularization profile for different values of $\rho$. 
Types of Regularization

- **Ridge regression / Tikhonov regularization**: $\rho = 2$ (Euclidean norm)
- **LASSO**: $\rho = 1$ (Manhattan norm)
- **Sparsity-inducing** regularization: $\rho \leq 1$ (nonconvex for $\rho < 1$)
- **Elastic net**\(^2\) regularization: mixed $L_1/L_2$ penalty:

\[
\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda \left[ (1 - \alpha)\|\theta\|_2^2 + \alpha \|\theta\|_1 \right]
\]

\(^1\)Figure source: http://scikit-learn.sourceforge.net
\(^2\)Also see Zou and Hastie, *Regularization and variable selection via the elastic net*
Regularized linear regression

A new loss or error function to minimize

\[
R_N(\theta, \theta_0) = \sum_{n} (y_n - \theta^T x_n - \theta_0)^2 + \lambda \|\theta\|_2^2
\]

where \( \lambda > 0 \) controls the model complexity, “shrinking” weights towards 0.

- If \( \lambda \to +\infty \), then
  \[
  \hat{\theta} \to 0
  \]

- If \( \lambda \to 0 \), back to normal OLS (Ordinary Least Squares).

For regularized linear regression: the solution changes very little (in form) from the OLS solution

\[
\arg\min \sum_{n} (y_n - \theta^T x_n - \theta_0)^2 + \lambda \|\theta\|_2^2 \Rightarrow \hat{\theta} = (X^T X + \lambda I)^{-1} X^T y
\]

and reduces to the OLS solution when \( \lambda = 0 \), as expected.

As long as \( \lambda \geq 0 \), the optimization problem remains convex.
Example: overfitting with polynomials

Our regression model

\[ y = \sum_{m=1}^{M} \theta_m x^m \]

Regularization would discourage large parameter values as we saw with the OLS solution, thus potentially preventing overfitting.

<table>
<thead>
<tr>
<th>( M = 0 )</th>
<th>( M = 1 )</th>
<th>( M = 3 )</th>
<th>( M = 9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \theta_0 )</td>
<td>0.19</td>
<td>0.82</td>
<td>0.31</td>
</tr>
<tr>
<td>( \theta_1 )</td>
<td>-1.27</td>
<td>7.99</td>
<td>232.37</td>
</tr>
<tr>
<td>( \theta_2 )</td>
<td>-25.43</td>
<td>-5321.83</td>
<td></td>
</tr>
<tr>
<td>( \theta_3 )</td>
<td>17.37</td>
<td>48568.31</td>
<td></td>
</tr>
<tr>
<td>( \theta_4 )</td>
<td></td>
<td>-231639.30</td>
<td></td>
</tr>
<tr>
<td>( \theta_5 )</td>
<td></td>
<td>640042.26</td>
<td></td>
</tr>
<tr>
<td>( \theta_6 )</td>
<td></td>
<td>-1061800.52</td>
<td></td>
</tr>
<tr>
<td>( \theta_7 )</td>
<td></td>
<td>1042400.18</td>
<td></td>
</tr>
<tr>
<td>( \theta_8 )</td>
<td></td>
<td>-557682.99</td>
<td></td>
</tr>
<tr>
<td>( \theta_9 )</td>
<td></td>
<td>125201.43</td>
<td></td>
</tr>
</tbody>
</table>
Overfitting in terms of $\lambda$

Overfitting is reduced from complex model to simpler one with the help of increasing regularizers

$\lambda$ vs. residual error shows the difference of the model performance on training and testing dataset
The effect of $\lambda$

Large $\lambda$ attenuates parameters towards 0

<table>
<thead>
<tr>
<th></th>
<th>$\ln \lambda = -\infty$</th>
<th>$\ln \lambda = -18$</th>
<th>$\ln \lambda = 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_0$</td>
<td>0.35</td>
<td>0.35</td>
<td>0.13</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>232.37</td>
<td>4.74</td>
<td>-0.05</td>
</tr>
<tr>
<td>$\theta_2$</td>
<td>-5321.83</td>
<td>-0.77</td>
<td>-0.06</td>
</tr>
<tr>
<td>$\theta_3$</td>
<td>48568.31</td>
<td>-31.97</td>
<td>-0.06</td>
</tr>
<tr>
<td>$\theta_4$</td>
<td>-231639.30</td>
<td>-3.89</td>
<td>-0.03</td>
</tr>
<tr>
<td>$\theta_5$</td>
<td>640042.26</td>
<td>55.28</td>
<td>-0.02</td>
</tr>
<tr>
<td>$\theta_6$</td>
<td>-1061800.52</td>
<td>41.32</td>
<td>-0.01</td>
</tr>
<tr>
<td>$\theta_7$</td>
<td>1042400.18</td>
<td>-45.95</td>
<td>-0.00</td>
</tr>
<tr>
<td>$\theta_8$</td>
<td>-557682.99</td>
<td>-91.53</td>
<td>0.00</td>
</tr>
<tr>
<td>$\theta_9$</td>
<td>125201.43</td>
<td>72.68</td>
<td>0.01</td>
</tr>
</tbody>
</table>
The effect of $\lambda$

Increasing $\lambda$ reduces variance (left) and increases bias (right)$^3$.

![Illustration of the dependence of bias and variance on model complexity, governed by a regularization parameter $\lambda$, using the sinusoidal data set from Chapter 1. There are $L = 100$ data sets, each having $N = 25$ data points, and there are $24$ Gaussian basis functions in the model so that the total number of parameters is $M = 25$ including the bias parameter. The left column shows the result of fitting the model to the data sets for various values of $\ln \lambda$ (for clarity, only 20 of the 100 fits are shown). The right column shows the corresponding average of the 100 fits (red) along with the sinusoidal function from which the data sets were generated (green).](image)

$^3$Bishop PRML Figure 3.5
$L_1$ promotes sparsity

$L_1$ regularization often leads to optimal solutions with many zeros, i.e., the regression function depends only on the (small) number of features with non-zero parameters.
Regularization in R demo

http://www.stats.ox.ac.uk/~palamara/teaching/SML19/regularization.html
What if $X^T X$ is not invertible?

Can you think of any reasons why that could happen?

**Answer 1:** $N < D$. Intuitively, not enough data to estimate all the parameters.

**Answer 2:** $X$ columns are not linearly independent. Intuitively, there are two features that are perfectly correlated. In this case, solution is not unique.
Ridge regression

**Intuition:** what does a non-invertible \( X^T X \) mean? Consider the SVD of this matrix:

\[
X^T X = V \begin{bmatrix}
\lambda_1 & 0 & 0 & \cdots & 0 \\
0 & \lambda_2 & 0 & \cdots & 0 \\
0 & \cdots & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & \lambda_r & 0 \\
0 & \cdots & \cdots & 0 & 0
\end{bmatrix} V^T
\]

where \( \lambda_1 \geq \lambda_2 \geq \cdots \lambda_r > 0 \) and \( r < D \).

**Regularization can fix this problem** by ensuring all singular values are non-zero

\[
X^T X + \lambda I = V \text{diag}(\lambda_1 + \lambda, \lambda_2 + \lambda, \cdots, \lambda) V^T
\]

where \( \lambda > 0 \) and \( I \) is the identity matrix
Computational complexity

Bottleneck of computing the solution? The OLS problem has a simple, closed-form solution. But computing it involves a number of matrix operations:

$$\theta = \left( X^T X \right)^{-1} X^T y$$

Matrix multiply of $X^T X \in \mathbb{R}^{(D+1) \times (D+1)}$
Inverting the matrix $X^T X$

How many operations do we need?

- $O(ND^2)$ for matrix multiplication
- $O(D^3)$ (e.g., using Gauss-Jordan elimination) or $O(D^{2.373})$ (recent theoretical advances) for matrix inversion
- Impractical for very large $D$ or $N$
- As an alternative, we could use numerical methods. This type of approach is widely used in several other machine learning algorithms. These methods are often the only available option, since sometimes we don’t have a closed form solution available.
Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\theta$ to $\theta^{(0)}$ (e.g., randomly); set $t = 0$; choose $\eta > 0$
- Loop until convergence
  1. Compute the gradient
     \[ \nabla R_N(\theta) = X^T \left( X \theta^{(t)} - y \right) \]
  2. Update the parameters
     \[ \theta^{(t+1)} = \theta^{(t)} - \eta \nabla R_N(\theta) \]
  3. $t \leftarrow t + 1$

What is the complexity of each iteration?
Gradient Descent

Start at a random point
Gradient Descent

Start at a random point

Determine a descent direction

\[ f(w) \]
Gradient Descent

Start at a random point
Determine a descent direction
Choose a step size

\[ f(w) \]

\[ w^* \]

\[ w_0 \]
Gradient Descent

Start at a random point

Determine a descent direction

Choose a step size

Update

\[ f(w) \]

\[ w^* \]

\[ w_1 \]

\[ w_0 \]
Gradient Descent

Start at a random point

**Repeat**  
- Determine a descent direction  
- Choose a step size  
- Update

**Until** stopping criterion is satisfied
Gradient Descent

Start at a random point

Repeat

- Determine a descent direction
- Choose a step size
- Update

Until stopping criterion is satisfied

\[ f(w) \]

- \[ w^* \]
- \[ w_1 \]
- \[ w_0 \]
Gradient Descent

Start at a random point

Repeat

- Determine a descent direction
- Choose a step size
- Update

Until stopping criterion is satisfied
Gradient Descent

Start at a random point

Repeat
  Determine a descent direction
  Choose a step size
  Update

Until stopping criterion is satisfied
Gradient Descent

Start at a random point
Repeat
  Determine a descent direction
  Choose a step size
  Update
Until stopping criterion is satisfied
Gradient Descent

Start at a random point

**Repeat**
- Determine a descent direction
- Choose a step size
- Update

**Until** stopping criterion is satisfied

$f(w)$

\[ w^* \ldots w_2 \ w_1 \ w_0 \]
Gradient descent

$h_\theta(x) = -900 - 0.1x$

$R_N(\theta_1)$
Gradient descent

\[ h_\theta(x) \]

\[ R_N(\theta_1) \]
Gradient descent

\[ h_\theta(x) \]

\[ R_N(\theta_1) \]
Gradient descent

\[ h_\theta(x) \quad R_N(\theta_1) \]
Gradient descent

\[ h_\theta(x) \quad R_N(\theta_1) \]
Gradient descent

\[ h_\theta(x) \quad R_N(\theta_1) \]
Gradient descent

\[ h_\theta(x) \]

\[ R_N(\theta_1) \]
Gradient descent

\[ h_\theta(x) \]

\[ R_N(\theta_1) \]
Gradient descent

\[ h_\theta(x) \]

\[ R_N(\theta_1) \]
Seeing in action

Choosing the right $\eta$ is important

small $\eta$ is too slow?  

large $\eta$ is too unstable?

To see if gradient descent is working, print out function value at each iteration.

- The value should decrease at each iteration.
- Otherwise, adjust $\eta$.  

Stochastic gradient descent

**Widrow-Hoff rule**: update parameters using one example at a time

- Initialize $\theta$ to $\theta^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop until convergence
  1. randomly choose training sample $x_t$
  2. Compute its contribution to the gradient
     $$g_t = (x_t^T \theta^{(t)} - y_t)x_t$$
  3. Update the parameters
     $$\theta^{(t+1)} = \theta^{(t)} - \eta g_t$$
  4. $t \leftarrow t + 1$

How does the complexity per iteration compare with gradient descent?
Gradient descent: mini-summary

- Batch gradient descent computes the exact gradient.
- Stochastic gradient descent approximates the gradient with a single data point; Its expectation equals the true gradient.
- Mini-batch variant: trade-off between accuracy of estimating gradient and computational cost
- Similar ideas extend to other ML optimization problems.
  - For large-scale problems, stochastic gradient descent often works well.
Classification
Suppose we made a prediction \( \hat{Y} = f(X) \in \mathcal{Y} \) based on observation of \( X \).

How good is the prediction? We can use a **loss function** \( L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+ \) to formalize the quality of the prediction.

**Typical loss functions:**
- **Squared loss** for regression
  \[
  L(Y, f(X)) = (f(X) - Y)^2 .
  \]
- **Absolute loss** for regression
  \[
  L(Y, f(X)) = |f(X) - Y| .
  \]
- **Misclassification loss** (or **0-1 loss**) for classification
  \[
  L(Y, f(X)) = \begin{cases} 
  0 & f(X) = Y \\
  1 & f(X) \neq Y 
  \end{cases} .
  \]

Many other choices are possible, e.g., **weighted misclassification loss**.

In classification, if estimated probabilities \( \hat{p}(k) \) for each class \( k \in \mathcal{Y} \) are returned, **log-likelihood loss** (or **log loss**) \( L(Y, \hat{p}) = - \log \hat{p}(Y) \) is often used.
The Bayes Classifier

What is the optimal classifier if the joint distribution \((X, Y)\) were known?

The density \(g\) of \(X\) can be written as a mixture of \(K\) components (corresponding to each of the classes):

\[
g(x) = \sum_{k=1}^{K} \pi_k g_k(x),
\]

where, for \(k = 1, \ldots, K\),

- \(\mathbb{P}(Y = k) = \pi_k\) are the class probabilities,
- \(g_k(x)\) is the conditional density of \(X\), given \(Y = k\).

The **Bayes classifier** \(f_{\text{Bayes}} : x \mapsto \{1, \ldots, K\}\) is the one with minimum risk:

\[
R(f) = \mathbb{E} [L(Y, f(X))] = \mathbb{E}_X \left[ \mathbb{E}_{Y|X} [L(Y, f(X))|X] \right]
\]

\[
= \int_X \mathbb{E} [L(Y, f(X))|X = x] g(x) dx
\]

The minimum risk attained by the Bayes classifier is called **Bayes risk**.

Minimizing \(\mathbb{E}[L(Y, f(X))|X = x]\) separately for each \(x\) suffices.
The Bayes Classifier

- Consider the 0-1 loss.
- The risk simplifies to:

\[
E \left[ L(Y, f(X)) \mid X = x \right] = \sum_{k=1}^{K} L(k, f(x)) \mathbb{P}(Y = k \mid X = x) \\
= 1 - \mathbb{P}(Y = f(x) \mid X = x)
\]

- The risk is minimized by choosing the class with the greatest probability given the observation:

\[
f_{\text{Bayes}}(x) = \arg \max_{k=1,\ldots,K} \mathbb{P}(Y = k \mid X = x)
\]

\[
= \arg \max_{k=1,\ldots,K} \frac{\pi_k g_k(x)}{\sum_{j=1}^{K} \pi_j g_j(x)} = \arg \max_{k=1,\ldots,K} \pi_k g_k(x).
\]

- The functions \( x \mapsto \pi_k g_k(x) \) are called **discriminant functions**. The discriminant function with maximum value determines the predicted class of \( x \).
The Bayes Classifier: Example

A simple two Gaussians example: Suppose $X \sim \mathcal{N}(\mu_Y, 1)$, where $\mu_1 = -1$ and $\mu_2 = 1$ and assume equal class probabilities $\pi_1 = \pi_2 = 1/2$.

\[ g_1(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x + 1)^2}{2}\right) \quad \text{and} \quad g_2(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x - 1)^2}{2}\right). \]

Optimal classification is $f_{\text{Bayes}}(x) = \arg \max_{k=1,\ldots,K} \pi_k g_k(x) = \begin{cases} 1 & \text{if } x < 0, \\ 2 & \text{if } x \geq 0. \end{cases}$
The Bayes Classifier: Example

How do you classify a new observation $x$ if now the standard deviation is still 1 for class 1 but $1/3$ for class 2?

Looking at density in a log-scale, optimal classification is to select class 2 if and only if $x \in [0.34, 2.16]$. 
Plug-in Classification

- The Bayes Classifier:

\[ f_{\text{Bayes}}(x) = \arg \max_{k=1,\ldots,K} \pi_k g_k(x). \]

- We know neither the conditional densities \( g_k \) nor the class probabilities \( \pi_k \)!
- The **plug-in classifier** chooses the class

\[ f(x) = \arg \max_{k=1,\ldots,K} \hat{\pi}_k \hat{g}_k(x), \]

where we plugged in

  - estimates \( \hat{\pi}_k \) of \( \pi_k \) and \( k = 1, \ldots, K \) and
  - estimates \( \hat{g}_k(x) \) of conditional densities,

- **Linear Discriminant Analysis** is an example of plug-in classification.