Applied Machine Learning

Regularization

Siamak Ravanbakhsh

COMP 551 (winter 2020)
Basic idea of

- overfitting and underfitting
- Regularization (L1 & L2)
- MLE vs MAP estimation
- bias and variance trade off
- evaluation metrics & cross validation
Previously...

Linear regression and logistic regression
is linear too simple? what if it's not a good fit?
how to increase the models expressiveness?

• create new nonlinear features
• is there a downside?
Recall: nonlinear basis functions

replace original features in \( f_w(x) = \sum_d w_d x_d \)
Recall: nonlinear basis functions

replace original features in \( f_w(x) = \sum_d w_d x_d \)

with nonlinear bases \( f_w(x) = \sum_d w_d \phi_d(x) \)
Recall: nonlinear basis functions

replace original features in  \( f_w(x) = \sum_d w_d x_d \)
with nonlinear bases  \( f_w(x) = \sum_d w_d \phi_d(x) \)
linear least squares solution  \( w^* = (\Phi^\top \Phi)^{-1} \Phi^\top y \)
replacing  \( X \) with  \( \Phi \)

\[
\Phi = \begin{bmatrix}
\phi_1(x^{(1)}), & \phi_2(x^{(1)}), & \cdots, & \phi_D(x^{(1)}) \\
\phi_1(x^{(2)}), & \phi_2(x^{(2)}), & \cdots, & \phi_D(x^{(2)}) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_1(x^{(N)}), & \phi_2(x^{(N)}), & \cdots, & \phi_D(x^{(N)})
\end{bmatrix}
\]
Recall: nonlinear basis functions

replace original features in \( f_w(x) = \sum_d w_d x_d \)
with nonlinear bases \( f_w(x) = \sum_d w_d \phi_d(x) \)
linear least squares solution \( w^* = (\Phi^\top \Phi)^{-1} \Phi^\top y \)
replacing \( X \) with \( \Phi \)

a (nonlinear) feature

\[
\Phi = \begin{bmatrix}
\phi_1(x^{(1)}), & \phi_2(x^{(1)}), & \cdots, & \phi_D(x^{(1)}) \\
\phi_1(x^{(2)}), & \phi_2(x^{(2)}), & \cdots, & \phi_D(x^{(2)}) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_1(x^{(N)}), & \phi_2(x^{(N)}), & \cdots, & \phi_D(x^{(N)})
\end{bmatrix}
\]

one instance
Recall: nonlinear basis functions

Examples: original input is scalar $x \in \mathbb{R}$

- Polynomial bases
  $$\phi_k(x) = x^k$$

- Gaussian bases
  $$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$

- Sigmoid bases
  $$\phi_k(x) = \frac{1}{1 + e^{-\frac{x-\mu_k}{s}}}$$
Example: Gaussian bases

\[ \phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}} \]
Example: Gaussian bases

\[ \phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}} \]

\[ y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon \]
**Example:** Gaussian bases

\[ \phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}} \]

\[ y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon \]

our fit to data using 10 Gaussian bases
**Example: Gaussian bases**

\[
\phi_k(x) = e^{-\frac{(x - \mu_k)^2}{s^2}}
\]

\[
y^{(n)} = \sin(x^{(n)}) + \cos(\sqrt{|x^{(n)}|}) + \epsilon
\]

Our fit to data using 10 Gaussian bases.

Prediction for a new instance:

\[
f(x') = \phi(x')^\top (\Phi^\top \Phi)^{-1} \Phi^\top y
\]

\(\mathbf{w}\) found using LLS.

New instance features evaluated for the new point.
Example: Gaussian bases

\[ \phi_k(x) = e^{\frac{(x-\mu_k)^2}{s^2}} \]

\[
\begin{align*}
\text{mu} &= \text{np.linspace}(0, 10, 10) & \text{#10 Gaussians bases} \\
\phi &= \lambda x, \mu: \text{np.exp}(-(x-\mu)^2) \\
\Phi &= \phi(x[:,None], \mu[None,:]) & \text{#N x 10} \\
w &= \text{np.linalg.lstsq}(\Phi, y)[0] & \text{#N x 10} \\
yh &= \text{np.dot}(\Phi, w) \\
\text{plt.plot}(x, y, 'b.') & \quad \text{why not more?}
\end{align*}
\]

plt.plot(x, yh, 'g-')
Example: Gaussian bases

\[ \phi_k(x) = e^{-\frac{(x - \mu_k)^2}{s^2}} \]

\[ \mu = \text{np.linspace}(0, 10, 50) \quad \# 50 \text{ Gaussians bases} \]

\[ \Phi = \phi(x[:, None], \mu[None, :]) \quad \# N \times 10 \]

\[ w = \text{np.linalg.lstsq}(\Phi, y)[0] \]

\[ yh = \text{np.dot}(\Phi, w) \]

\[ \text{plt.plot}(x, y, 'b.') \]

\[ \text{plt.plot}(x, yh, 'g-') \]
Example: Gaussian bases

\[ \phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}} \]

using 200, thinner bases (s=.1)

```python
#x: N
#y: N

mu = np.linspace(0, 10, 200)  #200 Gaussians bases

phi = lambda x, mu: np.exp(-((x-mu)/.1)**2)  

Phi = phi(x[:,None], mu[None,:]) #N x 10

w = np.linalg.lstsq(Phi, y)[0]  

yh = np.dot(Phi,w)

plt.plot(x, y, 'b.')
plt.plot(x, yh, 'g-')
```
**Example:** Gaussian bases

\[ \phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}} \]

Cost function \( J(w) \) is small and we have a "perfect" fit!

```python
# N
plt.plot(x, y, 'b.')
# N x 10
Phi = phi(x[:,None], mu[None,:])
w = np.linalg.lstsq(Phi, y)[0]
yh = np.dot(Phi,w)
plt.plot(x, yh, 'g-')
```
Generalization

which one of these models performs better at test time?

lower training error
Overfitting

which one of these models performs better at test time?

predictions of 4 models for the same input $f(x')$
Overfitting

which one of these models performs better at test time?

predictions of 4 models for the same input $f(x')$
Overfitting

which one of these models performs better at test time?

predictions of 4 models for the same input $f(x')$

- $D = 5$
- $D = 10$
- $D = 50$
- $D = 200$

underfitting

lowest test error

overfitting
Model selection

how to pick the model with lowest expected loss / test error?

**regularization** bound the test error by bounding
- training error
- model complexity
Model selection

how to pick the model with lowest expected loss / test error?

**regularization** bound the test error by bounding

- training error
- model complexity

use a **validation set** (and a separate test set for final assessment)

<table>
<thead>
<tr>
<th>Train</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>use for model selection</td>
<td>use for model selection</td>
<td>use for final model assessment</td>
</tr>
</tbody>
</table>
Model selection

how to pick the model with lowest expected loss / test error?

**regularization** bound the test error by bounding
- training error
- model complexity

use a **validation set** (and a separate test set for final assessment)

<table>
<thead>
<tr>
<th>Train</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>use for model selection</td>
<td>use for final model assessment</td>
<td></td>
</tr>
</tbody>
</table>
An observation

when overfitting, we often see large weights

dashed lines are $w_d \phi_d(x) \forall d$
An observation

when overfitting, we often see large weights

\[ w_d \phi_d(x) \quad \forall d \]

\[ D = 10 \]
An observation

when overfitting, we often see large weights

\[
dashed \text{ lines are } \ w_d \phi_d(x) \quad \forall d
\]
An observation

when overfitting, we often see large weights

\[ w_d \phi_d(x) \quad \forall d \]

\[ D = 10 \quad D = 15 \quad D = 20 \]
An observation

when overfitting, we often see large weights

dashed lines are $w_d \phi_d(x) \quad \forall d$

idea: penalize large parameter values
Ridge regression

L2 regularized linear least squares regression:

\[ J(w) = \frac{1}{2} \|Xw - y\|^2_2 + \frac{\lambda}{2} \|w\|^2_2 \]
Ridge regression

L2 regularized linear least squares regression:

\[ J(w) = \frac{1}{2} \| Xw - y \|_2^2 + \frac{\lambda}{2} \| w \|_2^2 \]

- sum of squared error
- \[ \frac{1}{2} \sum_n (y^{(n)} - w^\top x)^2 \]
Ridge regression

L2 regularized linear least squares regression:

$$J(w) = \frac{1}{2} \| Xw - y \|^2_2 + \frac{\lambda}{2} \| w \|^2_2$$

- sum of squared error
  $$\frac{1}{2} \sum_n (y^{(n)} - w^\top x)^2$$
- (squared) L2 norm of $$w$$
  $$w^T w = \sum_d w^2$$
Ridge regression

L2 regularized linear least squares regression:

\[ J(w) = \frac{1}{2} \| Xw - y \|_2^2 + \frac{\lambda}{2} \| w \|_2^2 \]

- sum of squared error
  \[ \frac{1}{2} \sum_n (y^{(n)} - w^\top x)^2 \]
- (squared) L2 norm of \( w \)
  \[ w^T w = \sum_d w^2 \]

regularization parameter \( \lambda > 0 \) controls the strength of regularization
Ridge regression

L2 regularized linear least squares regression:

\[ J(w) = \frac{1}{2} \left\| Xw - y \right\|_2^2 + \frac{\lambda}{2} \left\| w \right\|_2^2 \]

- sum of squared error
- (squared) L2 norm of \( w \)

regularization parameter \( \lambda > 0 \) controls the strength of regularization

a good practice is to not penalize the intercept \( \lambda(\left\| w \right\|_2^2 - w_0^2) \)
Ridge regression

we can set the derivative to zero

\[ J(w) = \frac{1}{2} (Xw - y)^\top (Xw - y) + \frac{\lambda}{2} w^\top w \]

\[ \nabla J(w) = X^\top (Xw - y) + \lambda w = 0 \]
Ridge regression

we can set the derivative to zero

\[ J(w) = \frac{1}{2} (Xw - y)^\top (Xw - y) + \lambda w^\top w \]

\[ \nabla J(w) = X^\top (Xw - y) + \lambda w = 0 \]

when using gradient descent, this term reduces the weights at each step (weight decay)
Ridge regression

we can set the derivative to zero

\[ J(w) = \frac{1}{2} (Xw - y)^\top (Xw - y) + \frac{1}{2} w^\top w \]
\[ \nabla J(w) = X^\top (Xw - y) + \lambda w = 0 \]

\[ (X^\top X + \lambda I)w = X^\top y \]

when using gradient descent, this term reduces the weights at each step (weight decay)
Ridge regression

we can set the derivative to zero

\[
J(w) = \frac{1}{2}(Xw - y)^\top (Xw - y) + \frac{\lambda}{2} w^\top w
\]

\[
\nabla J(w) = X^\top (Xw - y) + \lambda w = 0
\]

\[
(X^\top X + \lambda I)w = X^\top y
\]

\[
w = (X^\top X + \lambda I)^{-1} X^\top y
\]

when using gradient descent, this term reduces the weights at each step (weight decay)
Ridge regression

we can set the derivative to zero

\[ J(w) = \frac{1}{2}(Xw - y)^\top(Xw - y) + \frac{\lambda}{2}w^\top w \]

\[ \nabla J(w) = X^\top(Xw - y) + \lambda w = 0 \]

\[ (X^\top X + \lambda I)w = X^\top y \]

\[ w = (X^\top X + \lambda I)^{-1} X^\top y \]

the only part different due to regularization

when using gradient descent, this term reduces the weights at each step (**weight decay**)
Ridge regression

we can set the derivative to zero

\[
J(w) = \frac{1}{2} (Xw - y)^\top (Xw - y) + \frac{\lambda}{2} w^\top w
\]

\[
\nabla J(w) = X^\top (Xw - y) + \lambda w = 0
\]

\[
(X^\top X + \lambda I)w = X^\top y
\]

\[
w = (X^\top X + \lambda I)^{-1} X^\top y
\]

the only part different due to regularization

\(\lambda I\) makes it invertible!

we can have linearly dependent features (e.g., \(D > N\))

the solution will be unique!
**Example:** polynomial bases

\[ \phi_k(x) = x^k \]

Without regularization:

- using D=10 we can perfectly fit the data (high test error)
Example: polynomial bases

\[ \phi_k(x) = x^k \]

Without regularization:

- using D=10 we can perfectly fit the data (high test error)
Example: polynomial bases

\[ \phi_k(x) = x^k \]

Without regularization:

- using D=10 we can perfectly fit the data (high test error)
Example: polynomial bases

polynomial bases

$$\phi_k(x) = x^k$$

Without regularization:

- using $D=10$ we can perfectly fit the data (high test error)
Example: polynomial bases

$\phi_k(x) = x^k$

with regularization:

- fixed $D=10$, changing the amount of regularization
Example: polynomial bases

\[ \phi_k(x) = x^k \]

with regularization:

- fixed D=10, changing the amount of regularization
Example: polynomial bases

\[ \phi_k(x) = x^k \]

with regularization:

- fixed $D=10$, changing the amount of regularization
**Example: polynomial bases**

Polynomial bases

$$\phi_k(x) = x^k$$

With regularization:

- fixed $D=10$, changing the amount of regularization

![Graphs showing the effect of regularization with different values of $\lambda$.](image)
Data normalization

what if we scale the input features, using different factors

\[ \tilde{x}^{(n)} = \gamma_d x^{(n)} \forall d, n \]
Data normalization

what if we scale the input features, using different factors $\tilde{x}^{(n)} = \gamma_d x^{(n)} \forall d, n$

if we have no regularization: $\tilde{w}_d = \frac{1}{\gamma_d} w_d \forall d$

everything remains the same because: $||Xw - y||^2 = ||\tilde{X}\tilde{w} - y||^2$
Data normalization

what if we scale the input features, using different factors \( \tilde{x}^{(n)} = \gamma_d x^{(n)} \forall d, n \)

if we have no regularization: \( \tilde{w}_d = \frac{1}{\gamma_d} w_d \forall d \)

everything remains the same because: \[ \| Xw - y \|_2^2 = \| \tilde{X} \tilde{w} - y \|_2^2 \]

with regularization: \[ \| \tilde{w} \|_2 \neq \| w \|_2^2 \] so the optimal \( w \) will be different!
Data normalization

what if we scale the input features, using different factors \( \tilde{x}^{(n)} = \gamma_d x^{(n)} \forall d, n \)

if we have no regularization:  
\[
\tilde{w}_d = \frac{1}{\gamma_d} w_d \forall d
\]

everything remains the same because:
\[
||Xw - y||_2^2 = ||\tilde{X}\tilde{w} - y||_2^2
\]

with regularization:  
\[
||\tilde{w}||_2 \neq ||w||_2^2
\]

so the optimal \( w \) will be different!

features of different mean and variance will be penalized differently

\[
\left\{
\begin{aligned}
\mu_d &= \frac{1}{N} x_d^{(n)} \\
\sigma_d^2 &= \frac{1}{N-1} (x_d^{(n)} - \mu_d)^2
\end{aligned}
\right.
\]

makes sure all features have the same mean and variance
\[
x_d^{(n)} \leftarrow \frac{x_d^{(n)} - \mu_d}{\sigma_d}
\]
Maximum likelihood

previously: linear regression & logistic regression maximize log-likelihood
Maximum likelihood

previously: linear regression & logistic regression maximize log-likelihood

linear regression

\[ w^* = \arg \max_w p(y|w) \]
\[ = \arg \max_w \prod_{n=1}^N \mathcal{N}(y; \Phi w, \sigma^2) \]
\[ \equiv \arg \min \sum_n L_2(y^{(n)}, w^\top \phi(x^{(n)})) \]
Maximum likelihood

Previously: linear regression & logistic regression maximize log-likelihood

Linear regression

\[ w^* = \text{arg max}_w p(y|w) \]
\[ = \text{arg max}_w \prod_{n=1}^{N} \mathcal{N}(y; \Phi w, \sigma^2) \]
\[ \equiv \text{arg min} \sum_n L_2(y^{(n)}, w^\top \phi(x^{(n)})) \]

Logistic regression

\[ w^* = \text{arg max}_w p(y|x, w) \]
\[ = \text{arg max}_w \prod_{n=1}^{N} \text{Bernoulli}(y; \sigma(\Phi w)) \]
\[ \equiv \text{arg min} \sum_n L_{CE}(y^{(n)}, \sigma(w^\top \phi(x^{(n)}))) \]
Maximum likelihood

**previously:** linear regression & logistic regression maximize log-likelihood

**linear regression**

\[ w^* = \arg \max_p p(y|w) \]

\[ = \arg \max_w \prod_{n=1}^N \mathcal{N}(y; \Phi w, \sigma^2) \]

\[ \equiv \arg \min \sum_n L_2(y^{(n)}, w^T \phi(x^{(n)})) \]

**logistic regression**

\[ w^* = \arg \max_p p(y|x, w) \]

\[ = \arg \max_w \prod_{n=1}^N \text{Bernoulli}(y; \sigma(\Phi w)) \]

\[ \equiv \arg \min \sum_n L_{CE}(y^{(n)}, \sigma(w^T \phi(x^{(n)}))) \]

**idea:** maximize the posterior instead of likelihood

\[ p(w|y) = \frac{p(w)p(y|w)}{p(y)} \]
Maximum a Posteriori (MAP)

use the Bayes rule and find the parameters with max posterior prob.

\[ p(w|y) = \frac{p(w)p(y|w)}{p(y)} \]  the same for all choices of w (ignore)
Maximum a Posteriori (MAP)

use the Bayes rule and find the parameters with max posterior prob.

\[ p(w|y) = \frac{p(w)p(y|w)}{p(y)} \]

MAP estimate

\[ w^* = \arg \max_w p(w)p(y|w) \]
Maximum a Posteriori (MAP)

use the Bayes rule and find the parameters with max posterior prob.

\[ p(w|y) = \frac{p(w)p(y|w)}{p(y)} \]

MAP estimate

\[ w^* = \arg\max_w p(w)p(y|w) \]

\[ \equiv \arg\max_w \log p(y|w) + \log p(w) \]
Maximum a Posteriori (MAP)

use the Bayes rule and find the parameters with max posterior prob.

\[ p(w|y) = \frac{p(w)p(y|w)}{p(y)} \]

MAP estimate

\[ w^* = \arg \max_w p(w)p(y|w) \]

\[ \equiv \arg \max_w \log p(y|w) + \log p(w) \]

likelihood: original objective
**Maximum a Posteriori (MAP)**

use the Bayes rule and find the parameters with max posterior prob.

\[ p(w | y) = \frac{p(w)p(y|w)}{p(y)} \]

MAP estimate

\[ w^* = \arg \max_w p(w)p(y|w) \]

\[ \equiv \arg \max_w \left( \log p(y|w) + \log p(w) \right) \]

likelihood: original objective  prior
Maximum a Posteriori (MAP)

use the Bayes rule and find the parameters with max posterior prob.

\[ p(w|y) = \frac{p(w)p(y|w)}{p(y)} \]

MAP estimate

\[ w^* = \arg \max_w p(w)p(y|w) \]

\[ \equiv \arg \max_w \log p(y|w) + \log p(w) \]

likelihood: original objective  prior

even better would be to estimate the posterior distribution \( p(w|y) \)

• more on this later in the course!
Gaussian prior

Gaussian likelihood and Gaussian prior

\[ w^* = \arg \max_w p(w)p(y|w) \]
Gaussian prior

Gaussian likelihood and Gaussian prior

\[ w^* = \arg \max_w p(w) p(y|w) \equiv \arg \max_w \log p(y|w) + \log p(w) \]
Gaussian prior

Gaussian likelihood and Gaussian prior

\[ w^* = \arg \max_w p(w) p(y|w) \equiv \arg \max_w \log p(y|w) + \log p(w) \]

\[ \equiv \arg \max_w \log N(y|w^T x, \sigma^2) + \sum_{d=1}^D \log N(w_d, 0, \tau^2) \]
Gaussian prior

**Gaussian** likelihood and **Gaussian** prior

\[
 w^* = \arg \max_w p(w) p(y|w) = \arg \max_w \log p(y|w) + \log p(w) \\
\equiv \arg \max_w \log \mathcal{N}(y|w^\top x, \sigma^2) + \sum_{d=1}^{D} \log \mathcal{N}(w_d, 0, \tau^2) \text{ assuming independent Gaussian} \\
\text{(one per each weight)}
\]
Gaussian prior

Gaussian likelihood and Gaussian prior

\[ w^* = \arg \max_w \log p(y|w) + \log p(w) \]

\[ \equiv \arg \max_w \log \mathcal{N}(y|w^\top x, \sigma^2) + \sum_{d=1}^{D} \log \mathcal{N}(w_d, 0, \tau^2) \] assuming independent Gaussian

(one per each weight)

\[ \equiv \arg \max_w \frac{1}{2\sigma^2} (y - w^\top x)^2 - \sum_{d=1}^{D} \frac{1}{2\tau^2} w_d^2 \]
**Gaussian prior**

**Gaussian** likelihood and **Gaussian** prior

\[
w^* = \arg \max_w p(w) p(y|w) \equiv \arg \max_w \log p(y|w) + \log p(w)
\]

\[
\equiv \arg \max_w \log \mathcal{N}(y|w^\top x, \sigma^2) + \sum_{d=1}^D \log \mathcal{N}(w_d, 0, \tau^2) \quad \text{assuming independent Gaussian (one per each weight)}
\]

\[
\equiv \arg \max_w \frac{1}{2\sigma^2} (y - w^\top x)^2 - \sum_{d=1}^D \frac{1}{2\tau^2} w_d^2 \equiv \arg \min_w \frac{1}{2} (y - w^\top x)^2 + \sum_{d=1}^D \frac{\sigma^2}{2\tau^2} w_d^2
\]
**Gaussian prior**

**Gaussian** likelihood and **Gaussian** prior

\[
\begin{align*}
    w^* &= \arg \max_w p(w) p(y|w) \\
    &= \arg \max_w \log p(y|w) + \log p(w) \\
    &\equiv \arg \max_w \log \mathcal{N}(y|w^\top x, \sigma^2) + \sum_{d=1}^{D} \log \mathcal{N}(w_d, 0, \tau^2) \quad \text{assuming independent Gaussian (one per each weight)} \\
    &\equiv \arg \max_w -\frac{1}{2\sigma^2} (y - w^\top x)^2 - \sum_{d=1}^{D} \frac{1}{2\tau^2} w_d^2 \\
    &\equiv \arg \min_w \frac{1}{2} (y - w^\top x)^2 + \sum_{d=1}^{D} \frac{\sigma^2}{2\tau^2} w_d^2 \\
\end{align*}
\]

*multiple data-points*

\[
\begin{align*}
    &\equiv \arg \min_w \frac{1}{2} \sum_n (y^{(n)} - w^\top x^{(n)})^2 + \sum_{d=1}^{D} \frac{\lambda}{2} w_d^2
\end{align*}
\]
Gaussian prior

Gaussian likelihood and Gaussian prior

\[ w^* = \arg \max_w p(w)p(y|w) = \arg \max_w \log p(y|w) + \log p(w) \]

\[ = \arg \max_w \log \mathcal{N}(y|w^\top x, \sigma^2) + \sum_{d=1}^{D} \log \mathcal{N}(w_d, 0, \tau^2) \]

assuming independent Gaussian (one per each weight)

\[ = \arg \max_w \frac{1}{2\sigma^2} (y - w^\top x)^2 - \sum_{d=1}^{D} \frac{1}{2\tau^2} w_d^2 = \arg \min_w \frac{1}{2} (y - w^\top x)^2 + \sum_{d=1}^{D} \frac{\sigma^2}{2\tau^2} w_d^2 \]

multiple data-points

\[ \lambda = \frac{\sigma^2}{\tau^2} \]

L2 regularization
Gaussian prior

**Gaussian** likelihood and **Gaussian** prior

\[ w^* = \arg \max_w p(w)p(y|w) = \arg \max_w \log p(y|w) + \log p(w) \]

\[ \equiv \arg \max_w \log \mathcal{N}(y|w^\top x, \sigma^2) + \sum_{d=1}^{D} \log \mathcal{N}(w_d, 0, \tau^2) \]

assuming independent Gaussian

(one per each weight)

\[ \equiv \arg \max_w \frac{1}{2\sigma^2}(y - w^\top x)^2 - \sum_{d=1}^{D} \frac{1}{2\tau^2} w_d^2 \equiv \arg \min_w \frac{1}{2}(y - w^\top x)^2 + \sum_{d=1}^{D} \frac{\sigma^2}{2\tau^2} w_d^2 \]

multiple data-points

\[ \lambda = \frac{\sigma^2}{\tau^2} \]

\[ \equiv \arg \min_w \frac{1}{2} \sum_n (y^{(n)} - w^\top x^{(n)})^2 + \sum_{d=1}^{D} \frac{\lambda}{2} w_d^2 \]

L2 regularization

L2- regularization is assuming a Gaussian prior on weights

the same is true for logistic regression (or any other cost function)
Laplace prior

another notable choice of prior is the Laplace distribution
another notable choice of prior is the Laplace distribution

\[ p(w; \beta) = \frac{1}{2\beta} e^{-\frac{|w|}{\beta}} \]

notice the peak around zero
another notable choice of prior is the Laplace distribution

minimizing negative log-likelihood

\[ -\sum_d \log p(w_d) = \sum_d \frac{1}{2\beta} |w_d| \]

\[ p(w; \beta) = \frac{1}{2\beta} e^{-\frac{|w|}{\beta}} \]

notice the peak around zero
Laplace prior

another notable choice of prior is the Laplace distribution

minimizing negative log-likelihood

\[-\sum_d \log p(w_d) = \sum_d \frac{1}{2\beta} |w_d| = \frac{1}{2\beta} \|w\|_1\]

L1 norm of w

\[p(w; \beta) = \frac{1}{2\beta} e^{-\frac{|w|}{\beta}}\] notice the peak around zero

image:https://stats.stackexchange.com/questions/177210/why-is-laplace-prior-producing-sparse-solutions
Laplace prior

another notable choice of prior is the Laplace distribution

minimizing negative log-likelihood

\[- \sum_d \log p(w_d) = \sum_d \frac{1}{2\beta} |w_d| = \frac{1}{2\beta} ||w||_1\]

L1 norm of w

L1 regularization: \( J(w) \leftarrow J(w) + \lambda||w||_1 \) also called lasso (least absolute shrinkage and selection operator)

\[ p(w; \beta) = \frac{1}{2\beta} e^{-\frac{|w|}{\beta}} \]

notice the peak around zero

image:https://stats.stackexchange.com/questions/177210/why-is-laplace-prior-producing-sparse-solutions
$L_1$ vs $L_2$ regularization

Regularization path shows how $\{w_d\}$ change as we change $\lambda$.
$L_1$ vs $L_2$ regularization

regularization path shows how $\{w_d\}$ change as we change $\lambda$

Lasso produces sparse weights (many are zero, rather than small)
$L_1 \text{ vs } L_2$ regularization

regularization path shows how $\{w_d\}$ change as we change $\lambda$

Lasso produces sparse weights (many are zero, rather than small)

red-line is the optimal $\lambda$ from cross-validation

---

Lasso

Ridge regression

---

decreasing regularization coef. $\lambda$
$L_1$ vs $L_2$ regularization

$$\min_w J(w) + \lambda ||w||_p^p$$

is equivalent to

$$\min_w J(w) \text{ subject to } ||w||_p^p \leq \tilde{\lambda}$$

for an appropriate choice of $\tilde{\lambda}$.
$L_1 \text{ vs } L_2$ regularization

\[
\min_w J(w) + \lambda \|w\|_p^p
\]
is equivalent to \( \min_w J(w) \text{ subject to } \|w\|_p^p \leq \tilde{\lambda} \) for an appropriate choice of \( \tilde{\lambda} \).

The figures below show the constraint and the isocontours of \( J(w) \).

\( \min_w J(w) + \lambda \|w\|_p^p \) is equivalent to \( \min_w J(w) \text{ subject to } \|w\|_p^p \leq \tilde{\lambda} \) for an appropriate choice of \( \tilde{\lambda} \).

figures below show the constraint and the isocontours of \( J(w) \).
$L_1$ vs $L_2$ regularization

$$\min_w J(w) + \lambda \|w\|^p_p$$

is equivalent to

$$\min_w J(w) \text{ subject to } \|w\|^p_p \leq \tilde{\lambda}$$

for an appropriate choice of $\tilde{\lambda}$

figures below show the constraint and the isocontours of $J(w)$

optimal solution with L1-regularization is more likely to have zero components
Subset selection

\[ \sum_d w_d^4 \quad \sum_d w_d^2 \quad \sum_d |w_d| \quad \sum_d |w_d|^{\frac{1}{2}} \quad \sum_d |w_d|^{\frac{1}{10}} \]
Subset selection

p-norms with $p \geq 1$ are convex (easier to optimize)
Subset selection

p-norms with $p \geq 1$ are convex (easier to optimize)

p-norms with $p \leq 1$ induces sparsity

\[
\begin{align*}
\sum_d w_d^4 & \quad \sum_d w_d^2 & \quad \sum_d |w_d| & \quad \sum_d |w_d|^{\frac{1}{2}} & \quad \sum_d |w_d|^{\frac{1}{10}}
\end{align*}
\]
Subset selection

p-norms with \( p \geq 1 \) are convex (easier to optimize)

\[ \sum_d w_d^4 \quad \sum_d w_d^2 \quad \sum_d |w_d| \quad \sum_d |w_d|^{1/2} \quad \sum_d |w_d|^{1/10} \]

\( L_0 \text{ norm} \) closer to \( 0\)-norm

p-norms with \( p \leq 1 \) induces sparsity
Subset selection

p-norms with $p \geq 1$ are convex (easier to optimize)

$p$-norms with $p \leq 1$ induces sparsity

$L_0$ norm penalizes the number of non-zero features

$$J(w) + \lambda \|w\|_0 = J(w) + \lambda \sum_d \mathbb{1}(w_d \neq 0)$$
Subset selection

- p-norms with $p \geq 1$ are convex (easier to optimize)
- p-norms with $p \leq 1$ induce sparsity

$\sum_d w_d^4$  $\sum_d w_d^2$  $\sum_d |w_d|$  $\sum_d |w_d|^{1/2}$  $\sum_d |w_d|^{1/10}$

$L_0$ norm penalizes the number of non-zero features

$J(w) + \lambda ||w||_0 = J(w) + \lambda \sum_d \mathbb{1}(w_d \neq 0)$

a penalty of $\lambda$ for each feature

performs feature selection
Subset selection

p-norms with $p \geq 1$ are convex (easier to optimize)

$p$-norms with $p \leq 1$ induces sparsity

$L_0$ norm closer to 0-norm

optimizing this is a difficult combinatorial problem:

- search over all $2^D$ subsets
Subset selection

p-norms with $p \geq 1$ are convex (easier to optimize)

- $\sum_d w_d^4$
- $\sum_d w_d^2$
- $\sum_d |w_d|$
- $\sum_d |w_d|^{\frac{1}{2}}$
- $\sum_d |w_d|^{\frac{1}{10}}$

p-norms with $p \leq 1$ induces sparsity

$L_0$ norm closer to 0-norm

optimizing this is a difficult combinatorial problem:
- search over all $2^D$ subsets

L1 regularization is a viable alternative to L0 regularization
Bias-variance decomposition for L2 loss
Bias-variance decomposition for L2 loss

assume a true distribution \( p(x, y) \)
Bias-variance decomposition

for L2 loss

assume a true distribution $p(x, y)$

the regression function is $f(x) = \mathbb{E}_p [y|x]$
Bias-variance decomposition for L2 loss

assume a true distribution \( p(x, y) \)

the regression function is \( f(x) = \mathbb{E}_p[y|x] \)

assume that a dataset \( D = \{(x^{(n)}, y^{(n)})\}_n \) is sampled from \( p(x, y) \)
Bias-variance decomposition

for L2 loss

assume a true distribution $p(x, y)$

the regression function is $f(x) = \mathbb{E}_p[y|x]$

assume that a dataset $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n}$ is sampled from $p(x, y)$

let $\hat{f}_\mathcal{D}$ be our model based on the dataset
Bias-variance decomposition

for L2 loss

assume a true distribution $p(x, y)$

the regression function is $f(x) = \mathbb{E}_p[y|x]$

assume that a dataset $D = \{(x^{(n)}, y^{(n)})\}_{n}$ is sampled from $p(x, y)$

let $\hat{f}_D$ be our model based on the dataset

what we care about is the expected loss (aka risk)

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

all blue items are random variables
Bias-variance decomposition

for L2 loss

what we care about is the expected loss (aka risk)

\[ \mathbb{E}[(\hat{f}_\mathcal{D}(x) - y)^2] \]
Bias-variance decomposition

what we care about is the expected loss (aka risk)

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

for L2 loss
Bias-variance decomposition

for L2 loss

what we care about is the expected loss (aka risk)

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

$$\hat{f}_D(x) + E[D(\hat{f}_D(x))] - E[D(\hat{f}_D(x))]$$

add and subtract a term
Bias-variance decomposition

for L2 loss

what we care about is the expected loss (aka risk)

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

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

add and subtract a term
Bias-variance decomposition

for L2 loss

what we care about is the expected loss *(aka risk)*

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

\[
= \mathbb{E}[(\hat{f}_D(x) - \mathbb{E}_D[\hat{f}_D(x)])^2] + \mathbb{E}[(f(x) - \mathbb{E}_D[\hat{f}_D(x)])^2] + \mathbb{E}[\epsilon^2]
\]

add and subtract a term

the remaining terms evaluate to zero (check for yourself!)
Bias-variance decomposition for L2 loss

what we care about is the expected loss (aka risk)

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

\[\hat{f}_D(x) + E_D[\hat{f}_D(x)] - E_D[\hat{f}_D(x)]\] add and subtract a term

the remaining terms evaluate to zero (check for yourself!)

\[
= E[(\hat{f}_D(x) - E_D[\hat{f}_D(x)])^2] + E[(f(x) - E_D[\hat{f}_D(x)])^2] + E[\epsilon^2]
\]

variance bias unavoidable noise error
Bias-variance decomposition
for L2 loss

the expected loss is decomposed to:

image: P. Domingos' posted article
Bias-variance decomposition for L2 loss

the expected loss is decomposed to:

$$\mathbb{E}[(f(x) - \mathbb{E}_D[\hat{f}_D(x)])^2]$$

**bias:** how average over all datasets differs from the regression function

image: P. Domingos' posted article
Bias-variance decomposition for L2 loss

the expected loss is decomposed to:

$$\mathbb{E}[(f(x) - \mathbb{E}_D[f_D(x)])^2]$$

**bias:** how average over all datasets differs from the regression function

$$\mathbb{E}[(\hat{f}_D(x) - \mathbb{E}_D[\hat{f}_D(x)])^2]$$

**variance:** how change of dataset affects the prediction

image: P. Domingos' posted article
Bias-variance decomposition for L2 loss

The expected loss is decomposed to:

\[ E[(f(x) - E_D[\hat{f}_D(x)])^2] \]

**Bias**: how average over all datasets differs from the regression function

\[ E[(\hat{f}_D(x) - E_D[\hat{f}_D(x)])^2] \]

**Variance**: how change of dataset affects the prediction

\[ E[\epsilon^2] \]

**Noise error**: the error even if we used the true model \( f(x) \)

Image: P. Domingos' posted article
Bias-variance decomposition for L2 loss

the expected loss is decomposed to:

$$\mathbb{E}[(f(x) - \mathbb{E}_D[\hat{f}_D(x)])^2]$$

**bias**: how average over all datasets differs from the regression function

$$\mathbb{E}[(\hat{f}_D(x) - \mathbb{E}_D[\hat{f}_D(x)])^2]$$

**variance**: how change of dataset affects the prediction

$$\mathbb{E}[\epsilon^2]$$

**noise error**: the error even if we used the true model $f(x)$

image: P. Domingos' posted article
Bias-variance decomposition

for L2 loss

the expected loss is decomposed to:

$$\mathbb{E}[(f(x) - \mathbb{E}_D[\hat{f}_D(x)])^2]$$

**bias:** how average over all datasets differs from the regression function

$$\mathbb{E}[(\hat{f}_D(x) - \mathbb{E}_D[\hat{f}_D(x)])^2]$$

**variance:** how change of dataset affects the prediction

$$\mathbb{E}[\epsilon^2]$$

**noise error:** the error even if we used the true model $f(x)$

different models vary in their trade off between error due to bias and variance

- simple models: often more biased
- complex models: often have more variance

image: P. Domingos' posted article
Example: bias vs. variance
Example: bias vs. variance

models for different datasets $\hat{f}_D$ using Gaussian bases

random datasets of size $N=25$ instances are not shown

$y$

$\ln \lambda = -0.31$

$x$

$0$

$1$

$0$

$-1$

$0$

$-1$

$t$

$0$

$1$

$0$

$-1$

$0$

$1$

$0$

$-1$
Example: bias vs. variance

Models for different datasets $\hat{f}_D$ using Gaussian bases. Random datasets of size $N=25$ instances are not shown.

True model $f$, their average $\mathbb{E}[\hat{f}_D]$.
Example: bias vs. variance

models for different datasets $\hat{f}_D$ using Gaussian bases

random datasets of size $N=25$ instances are not shown

true model $f$

their average $\mathbb{E}[\hat{f}_D]$

### Bias and Variance

- **Bias** is the difference (in L2 norm) between two curves and their average.
- **Variance** is the average difference (in squared L2 norm) between these curves and their average.

- $\ln \lambda = -0.31$
**Example:** bias vs. variance

![Diagram showing bias vs. variance](image.png)
Example: bias vs. variance

using larger regularization penalty: higher bias - lower variance
Example: bias vs. variance

using larger regularization penalty: higher bias - lower variance

side note
the average fit is very good, despite high variance

model averaging: uses "average" prediction of expressive models to prevent overfitting
Example: bias vs. variance
Example: bias vs. variance

the lowest expected loss (test error) is somewhere between the two extremes

Increasing bias  

Increasing variance
Example: bias vs. variance

The lowest expected loss (test error) is somewhere between the two extremes.

In reality, we don't have access to the true model. How to decide which model to use?
Big picture!

The graph shows the relationship between model complexity and prediction error. The x-axis represents model complexity, while the y-axis represents prediction error. The graph includes two main lines:

- **High Bias, Low Variance** line
- **Low Bias, High Variance** line

The graph illustrates how prediction error decreases as model complexity increases. The average training error and average test error are also indicated.

Error for random dataset $\mathcal{D}$ is also highlighted.
high variance in more complex models means that test and training error can be very different
Big picture!

high variance in more complex models means that test and training error can be very different

high bias in simplistic models means that training error can be high

$$\mathcal{D}$$
Model selection

how to pick the model with lowest expected loss / test error?

use a validation set (and a separate test set for final assessment)

- **Train** use for model selection
- **Validation** use for final model assessment
- **Test**

**regularization** bound the test error by bounding

- training error
- model complexity
Model selection

how to pick the model with lowest expected loss / test error?

use a **validation set** (and a separate test set for final assessment)

- **Train**
  - use for model selection
- **Validation**
  - use for model selection
- **Test**
  - use for final model assessment

**regularization** bound the test error by bounding
- training error
- model complexity

in the end we may have to use a validation set to find the right amount of regularization
Cross validation

getting a more reliable estimate of test error using validation set

K-fold cross validation (CV)

- randomly partition the data into $K$ folds
- use $K-1$ for training, and 1 for validation
- report average/std of the validation error over all folds
Cross validation

going a more reliable estimate of test error using validation set

K-fold cross validation (CV)

- randomly partition the data into K folds
- use K-1 for training, and 1 for validation
- report average/std of the validation error over all folds

leave-one-out CV: extreme case of k=N
Cross validation

getting a more reliable estimate of test error using validation set

K-fold cross validation (CV)

- randomly partition the data into k folds
- use k-1 for training, and 1 for validation
- report average/std of the validation error over all folds

image credit: Thanh Nguyen et al'19
Cross validation

getting a more reliable estimate of test error using validation set

K-fold cross validation (CV)

- randomly partition the data into k folds
- use k-1 for training, and 1 for validation
- report average/std of the validation error over all folds

once the hyper-parameters are selected, we can use the whole set for training
use test set for the final assessment

image credit: Thanh Nguyen et al'19
Evaluation

evaluation metric can be different from the optimization objective

confusion matrix is a CxC table that compares truth-vs-prediction

for binary classification:

<table>
<thead>
<tr>
<th>Result</th>
<th>Truth</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td></td>
<td>FN</td>
<td>TN</td>
</tr>
<tr>
<td>Σ</td>
<td>P</td>
<td>N</td>
</tr>
</tbody>
</table>
Evaluation

_evaluation metric_ can be different from the optimization objective

**confusion matrix** is a CxC table that compares truth-vs-prediction

for **binary classification**:

<table>
<thead>
<tr>
<th>Result</th>
<th>Truth</th>
<th>( \Sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>( \Sigma )</td>
<td>FN</td>
<td>TN</td>
</tr>
<tr>
<td>P</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

some _evaluation metrics_ (based on the confusion table)

\[
\text{Accuracy} = \frac{TP + TN}{P + N}
\]

\[
\text{Error rate} = \frac{FP + FN}{P + N}
\]

\[
\text{Precision} = \frac{TP}{RP}
\]

\[
\text{Recall} = \frac{TP}{P}
\]

\[
F_1\text{score} = 2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]
Evaluation

evaluation metric can be different from the optimization objective.

Confusion matrix is a CxC table that compares truth-vs-prediction.

For binary classification:

<table>
<thead>
<tr>
<th>Result</th>
<th>Truth</th>
<th>Σ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>FN</td>
<td>TN</td>
<td>RN</td>
</tr>
<tr>
<td>Σ</td>
<td>P</td>
<td>N</td>
</tr>
</tbody>
</table>

Some evaluation metrics (based on the confusion table):

- **Accuracy** = \( \frac{TP + TN}{P + N} \)
- **Error rate** = \( \frac{FP + FN}{P + N} \)
- **Precision** = \( \frac{TP}{RP} \)
- **Recall** = \( \frac{TP}{P} \)
- **F1 score** = \( \frac{2 \times Precision \times Recall}{Precision + Recall} \)

Type I vs Type II error
if we produce class score (probability)
we can trade-off between type I & type II error

\[ p(y = 1|x) \]

threshold
if we produce class score (probability)
we can trade-off between type I & type II error

\[ p(y = 1|x) \]

**goal:** evaluate class scores/probabilities (independent of choice of threshold)
Evaluation

if we produce class score (probability)
we can trade-off between type I & type II error

\[ p(y = 1| x) \]

\[ \text{threshold} \]

0 \hspace{1cm} 1

**goal:** evaluate class scores/probabilities (independent of choice of threshold)

Receiver Operating Characteristic **ROC curve**

**TPR** = TP/P (recall, sensitivity)

**FPR** = FP/N (fallout, false alarm)

\[ \text{TPR} = \frac{TP}{P} \]

\[ \text{FPR} = \frac{FP}{N} \]
Summary

- complex models can have very different training and test error (generalization gap)
- regularization bounds this gap by penalizing model complexity
  - L1 & L2 regularization
  - probabilistic interpretation: different priors on weights
  - L1 produces sparse solutions (useful for feature selection)
Summary

- complex models can have very different training and test error (*generalization gap*)
- regularization bounds this gap by penalizing model complexity
  - L1 & L2 regularization
  - probabilistic interpretation: different priors on weights
  - L1 produces sparse solutions (useful for feature selection)
- bias-variance trade off:
  - formalizes the relation between
    - training error (bias)
    - complexity (variance) and
    - and the test error (bias + variance)
  - not so elegant beyond L2 loss
Summary

- complex models can have very different training and test error (*generalization gap*)
- regularization bounds this gap by penalizing model complexity
  - L1 & L2 regularization
  - probabilistic interpretation: different priors on weights
  - L1 produces sparse solutions (useful for feature selection)
- bias-variance trade off:
  - formalizes the relation between
    - training error (bias)
    - complexity (variance) and
    - and the test error (bias + variance)
  - not so elegant beyond L2 loss
- (cross) validation for model selection