Machine Learning

Chao Lan
Machine Learning Prediction Models

Regression Model
- linear regression (least square, ridge regression, Lasso)

Classification Model
- naive Bayes, logistic regression, Gaussian discriminant analysis, k-nearest neighbor, linear support vector machine (LSVM), decision tree, neural network, (Bayesian network)
- multi-class classification with binary classifier

Advanced Model
- kernel, ensemble

Online Model
- SGD learner, HMM
Machine Learning Prediction Models

example X

feature vector

<table>
<thead>
<tr>
<th>google</th>
<th>lottery</th>
<th>cat</th>
<th>email</th>
<th>transport</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (x_1)</td>
<td>1 (x_2)</td>
<td>0 (x_3)</td>
<td>1 (x_4)</td>
<td>0 (x_5)</td>
<td>.</td>
</tr>
</tbody>
</table>

\[
[1 \ (x_1) \\
1 \ (x_2) \\
0 \ (x_3) \\
1 \ (x_4) \\
0 \ (x_5) \\
. \\
. \\
. \\
0 \ (x_p)]
\]

label Y

spam or, ham
Machine Learning Prediction Models

Regression Model
- linear regression (least square, ridge regression, Lasso)

Classification Model
- naive Bayes, logistic regression, Gaussian discriminant analysis, Bayesian network, k-Nearest Neighbor, linear support vector machine (LSVM), decision tree, neural network

Advanced Model
- kernel, ensemble

Online Model
- SGD learner, HMM
Linear Regression Model

Linear regression model has the following form

\[ y(x, w) = w_0 + w_1 x_1 + \ldots + w_D x_D \]

- w’s are regression coefficients
- w0 is the bias term
- y is label (e.g., how likely x is a spam)
Linear regression model has the following form:

\[ y(x, w) = w_0 + w_1 x_1 + \ldots + w_D x_D \]

- w's are regression coefficients
- w0 is the bias term
- y is label
  (e.g., how likely x is a spam)

**Feature vector**

\[
\begin{pmatrix}
\text{google} \\
\text{lottery}
\end{pmatrix} = \begin{pmatrix}
1 (x_1) \\
0 (x_2)
\end{pmatrix}
\]
Geometric Interpretation of the Bias Term

$w_0$ is the bias term. It allows the model to shift along the y axis (to better fit data).

$$y(x,w) = w_0 + w_1 x_1 + \ldots + w_D x_D$$
Three Learners of the Linear Regression Model

Three ways to learn $w$ in $y(x, w) = w_0 + w_1 x_1 + \ldots + w_D x_D$ from data.

1. least square

2. ridge regression

3. Lasso
Three Learners of the Linear Regression Model

Three ways to learn $w$ in  $y(x, w) = w_0 + w_1x_1 + \ldots + w_Dx_D$ from data.

1. least square: find $w$ that minimizes total squared error

2. ridge regression

3. Lasso

$$w^* = \arg\min_w \sum_{(x,y) \in D} [y(x, w) - y]^2$$
Three Learners of the Linear Regression Model

Three ways to learn $w$ in $y(x, w) = w_0 + w_1 x_1 + \ldots + w_D x_D$ from data.

1. least square: find $w$ that minimizes total squared error

2. ridge regression: find $w$ in a restricted range that minimizes total squared error

3. Lasso

$$w^* = \arg \min_w \sum_{(x, y) \in D} [y(x, w) - y]^2 + \lambda \sum_j w_j^2$$

How would $\lambda$ affect shrinkage?
Three Learners of the Linear Regression Model

Three ways to learn $\mathbf{w}$ in $y(x, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D$ from data.

1. least square: find $\mathbf{w}$ that minimizes the total squared error

2. ridge regression: find $\mathbf{w}$ in a restricted range that minimizes the total squared error

3. Lasso: find $\mathbf{w}$ with some elements eliminated that minimizes the total squared error

$$w^* = \arg \min_{\mathbf{w}} \sum_{(x,y) \in D} [y(x, \mathbf{w}) - y]^2 + \lambda \sum_j |w_j|$$

How would $\lambda$ affect elimination?
Coefficients Obtained by the Three Learners

<table>
<thead>
<tr>
<th>Term</th>
<th>LS</th>
<th>Best Subset</th>
<th>Ridge</th>
<th>Lasso</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept</td>
<td>2.465</td>
<td>2.477</td>
<td>2.452</td>
<td>2.468</td>
</tr>
<tr>
<td>lcavol</td>
<td>0.680</td>
<td>0.740</td>
<td>0.420</td>
<td>0.533</td>
</tr>
<tr>
<td>lweight</td>
<td>0.263</td>
<td>0.316</td>
<td>0.238</td>
<td>0.169</td>
</tr>
<tr>
<td>age</td>
<td>-0.141</td>
<td></td>
<td>-0.046</td>
<td></td>
</tr>
<tr>
<td>lbph</td>
<td>0.210</td>
<td></td>
<td>0.162</td>
<td>0.002</td>
</tr>
<tr>
<td>svi</td>
<td>0.305</td>
<td></td>
<td>0.227</td>
<td>0.094</td>
</tr>
<tr>
<td>lcp</td>
<td>-0.288</td>
<td></td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>gleason</td>
<td>-0.021</td>
<td></td>
<td>0.040</td>
<td></td>
</tr>
<tr>
<td>pgg45</td>
<td>0.267</td>
<td></td>
<td>0.133</td>
<td></td>
</tr>
<tr>
<td>Test Error</td>
<td>0.521</td>
<td>0.492</td>
<td>0.492</td>
<td>0.479</td>
</tr>
<tr>
<td>Std Error</td>
<td>0.179</td>
<td>0.143</td>
<td>0.165</td>
<td>0.164</td>
</tr>
</tbody>
</table>
Lasso: Automatically Eliminate Features by L1-Norm

\[
\min_w \sum_{(x,y) \in D} [y(x, w) - y]^2 + \lambda \sum_j w_j^2
\]

\[
\min_w \sum_{(x,y) \in D} [y(x, w) - y]^2 + \lambda \sum_j |w_j|
\]
Bias-Variance Tradeoff

Simple model often has small variance but large bias (in estimation).

Complex model often has small bias but large variance (a.k.a., overfitting).
Recap: Simple Model has Shrinked Range of $w$

A model with shrinked range of unknown parameters is usually simpler.

$$y(x, w) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$$E(w) = \frac{1}{2} \sum_{i=1}^{n} [y(x_i, w) - y_i]^2$$

$$\tilde{E}(w) = \frac{1}{2} \sum_{i=1}^{N} [y(x_i, w) - y_i]^2 + \lambda \cdot (w_0^2 + w_1^2 + \ldots + w_M^2)$$

$M = 9$

$\lambda \sim 10^{-8}$

$M = 9$

$\lambda = 1$
Recap: Larger $\lambda$ Learns a Simpler Model

$$E(w) = \frac{1}{2} \sum_{i=1}^{N} [(y(x_i, w) - y_i)^2 + \lambda \cdot (w_0^2 + w_1^2 + \ldots + w_M^2)]$$

<table>
<thead>
<tr>
<th>$w_i^*$</th>
<th>$\lambda = 0$</th>
<th>$\lambda = 10^{-8}$</th>
<th>$\lambda = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_0^*$</td>
<td>0.35</td>
<td>0.35</td>
<td>0.13</td>
</tr>
<tr>
<td>$w_1^*$</td>
<td>232.37</td>
<td>4.74</td>
<td>-0.05</td>
</tr>
<tr>
<td>$w_2^*$</td>
<td>-5321.83</td>
<td>-0.77</td>
<td>-0.06</td>
</tr>
<tr>
<td>$w_3^*$</td>
<td>48568.31</td>
<td>-31.97</td>
<td>-0.05</td>
</tr>
<tr>
<td>$w_4^*$</td>
<td>-231639.30</td>
<td>-3.89</td>
<td>-0.03</td>
</tr>
<tr>
<td>$w_5^*$</td>
<td>640042.26</td>
<td>55.28</td>
<td>-0.02</td>
</tr>
<tr>
<td>$w_6^*$</td>
<td>-1061800.52</td>
<td>41.32</td>
<td>-0.01</td>
</tr>
<tr>
<td>$w_7^*$</td>
<td>1042400.18</td>
<td>-45.95</td>
<td>-0.00</td>
</tr>
<tr>
<td>$w_8^*$</td>
<td>-557682.99</td>
<td>-91.53</td>
<td>0.00</td>
</tr>
<tr>
<td>$w_9^*$</td>
<td>125201.43</td>
<td>72.68</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Graphs showing $M = 9$ with $\lambda \sim 10^{-8}$ and $M = 9$ with $\lambda = 1$.
Bias-Variance Tradeoff

Simple model often has small variance but large bias (in estimation).

Complex model often has small bias but large variance (a.k.a., overfitting).
Bias-Variance Tradeoff

Simple model often has small variance but large bias \( (\text{in estimation}) \).

Complex model often has small bias but large variance \( (\text{a.k.a., overfitting}) \).
Bias-Variance Tradeoff

Simple model often has small variance but large bias \( \text{(in estimation)} \).

Complex model often has small bias but large variance \( \text{(a.k.a., overfitting)} \).
Bias-Variance Tradeoff

We also have theoretical evidence:

\[ er(x; f) = E[(f(x; S) - y)^2] \]

\[ = E[f(x; S)^2] + E[y^2] - 2E[f(x; S) \cdot y] \]

\[ = (Var[f(x; S)] + E[f(x; S)]^2) + (Var[y] + E[y]^2) - 2E[f(x; S)] \cdot E[y] \]

\[ = (E[f(x; S)]^2 - 2E[f(x; S)] \cdot E[y] + E[y]^2) + Var[f(x; S)] + Var[y] \]

\[ = (E[f(x; S)] - E[y])^2 + Var[f(x; S)] + Var[y]. \]

\[ = \text{bias}^2(f; x) + \text{variance}(f; x) + \text{variance}(y). \]
Regression Methods in Python

```python
import numpy as np
from sklearn import datasets
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

# Data Preparation
diabetes = datasets.load_diabetes()
X = diabetes.data[:, np.newaxis, 2]
X_train = X[:-20]
X_test = X[-20:]
y_train = diabetes.target[:-20]
y_test = diabetes.target[-20:]

# Model Construction (pick up of them)
# 1. Linear regression
model = linear_model.LinearRegression()
# 2. ridge regression
model = linear_model.Ridge(alpha=1e-1)
# 3. Lasso
model = linear_model.Lasso(alpha=1e-1)

# Model Training
model.fit(X_train, y_train)

# Model Testing/Assessment
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(mse)
```

\[ w^* = \arg \min_w \sum_{(x, y) \in D} [y(x, w) - y]^2 \]

\[ w^* = \arg \min_w \sum_{(x, y) \in D} [y(x, w) - y]^2 + \lambda \sum_j w_j^2 \]

\[ w^* = \arg \min_w \sum_{(x, y) \in D} [y(x, w) - y]^2 + \lambda \sum_j |w_j| \]
Machine Learning Prediction Models

Regression Model
- linear regression (least square, ridge regression, Lasso)

Classification Model
- naive Bayes, logistic regression, Gaussian discriminant analysis, k-nearest neighbor, linear support vector machine (LSVM), decision tree, neural network, (Bayesian network)
- multi-class classification with binary classifier

Advanced Model
- kernel, ensemble

Online Model
- SGD learner, HMM
A Probabilistic View of Classifier

A classifier can be a table of $p(y|x)$: given an example $x$, what is the distribution of label $y$?

- $p( y=\text{spam} | x=\text{blue email}) = 0.75$
- $p( y=\text{ham} | x=\text{blue email}) = 0.25$

- $p( y=\text{spam} | x=\text{red email}) = 0.4$
- $p( y=\text{ham} | x=\text{red email}) = 0.6$
1. Naive Bayes

Estimate $p(y|x)$ using (1) the Bayes’ rule and (2) the independent assumption of features.

$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$
Example of Bayes Classifier with One Feature

Goal: \( p( y = \text{spam} \mid x = \text{blue email} ) \)

- \( p( x = \text{blue email} \mid y = \text{spam} ) = \frac{1}{4} \)
- \( p( x = \text{blue email} ) = \frac{2}{6} = \frac{1}{3} \)
- \( p( y = \text{spam} ) = \frac{4}{6} = \frac{2}{3} \)
- \( p( y = \text{spam} \mid x = \text{blue email} ) = \frac{p( x = \text{blue email} \mid y = \text{spam} ) \times p( y = \text{spam} )}{p( x = \text{blue email} )} \)
Example of Naive Classifier

\[ p(x = \text{blue email} \& \text{“google”} \& \text{“lottery”} \mid y = \text{spam}) \]

\[ = p(x \text{ is blue} \mid y = \text{spam}) \]

* \[ p(x \text{ has word “google”} \mid y = \text{spam}) \]

* \[ p(x \text{ has word “lottery”} \mid y = \text{spam}) \]

Of course, we can also construct conditional probability for continuous variables.
```python
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:20]
X_test = X[20:]
y_train = iris.target[:20]
y_test = iris.target[20:]

# model construction
model = GaussianNB()

# model training
model.fit(X_train, y_train)

# model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
```
2. Gaussian Discriminant Analysis

Similar to Naive Bayes, but do not make i.i.d. assumption on features.

\[
P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}
\]

Assume \(p(x \mid y=k)\) is normal for each class \(k\) and estimate it from data.

\[
f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

Quadratic DA (QDA) assumes classes have different distributions. Linear DA (LDA) assumes covariance is shared by classes.
Example of GDA

\[ p(x = \text{blue email} \& \ "google" \& \ "lottery" \mid y = \text{spam}) \]

\[
 f_X(x_1, \ldots, x_k) = \frac{\exp\left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)}{\sqrt{(2\pi)^k |\Sigma|}}
\]

\[ x = \begin{pmatrix} \text{color} \\ \text{google} \\ \text{lottery} \end{pmatrix} \quad \mu = \begin{pmatrix} \text{color.m} \\ \text{google.m} \\ \text{lottery.m} \end{pmatrix} \quad \Sigma = \begin{pmatrix} \text{cov[c,c]}, \text{cov[c,g]}, \text{cov[c,l]} \\ \text{cov[g,c]}, \text{cov[g,g]}, \text{cov[g,l]} \\ \text{cov[l,c]}, \text{cov[l,g]}, \text{cov[l,l]} \end{pmatrix} \]

(\mu, \Sigma) can be estimated by MLE.

Quadratic DA (QDA) assumes different classes have different (\mu, \Sigma). Linear DA (LDA) assumes \Sigma is shared by all classes.
```python
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:20]
X_test = X[20:]
y_train = iris.target[:20]
y_test = iris.target[20:]

# model construction
model = LinearDiscriminantAnalysis()

# model training
model.fit(X_train, y_train)

# model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
```
3. Logistic Regression

Directly construct $p(y|x)$ with unknown parameters and estimate them by MLE.

$$P(y = 0 \mid x; \beta) = \frac{1}{1 + \exp(-x^T \beta)} = \frac{\exp(x^T \beta)}{1 + \exp(x^T \beta)}$$

$$P(y = 1 \mid x; \beta) = 1 - P(y = 1 \mid x; \beta) = \frac{1}{1 + \exp(x^T \beta)}$$

$$x^T \beta = x_1 \beta_1 + x_2 \beta_2 + \ldots + x_p \beta_p$$

$$L_n(\beta) = \log \prod_{i=1}^n P(y_i \mid x_i; \beta)$$

$$= \sum_{i=1}^n \log P(y_i \mid x_i; \beta)$$

$$= \sum_{i=1}^n (1 - y_i) (x_i^T \beta) - \log (1 + \exp(x_i^T \beta))$$

Logistic regression typically works with binary classification, i.e., $y = 0$ or $1$. And $\beta$ can be numerically solved by the Newton’s method.
Regularized Logistic Regression

It is logistic regression learned with restrictive range of parameter $\beta$.

- Larger $\lambda$ gives ______ model?

$$\beta^* = \arg \min_{\beta} -L(\beta) + \lambda \sum_j \beta_j^2$$

$$L_n(\beta) = \log \Pi_{i=1}^n P(y_i \mid x_i; \beta)$$

$$= \sum_{i=1}^n \log P(y_i \mid x_i; \beta)$$

$$= \sum_{i=1}^n (1 - y_i)(x_i^T \beta) - \log(1 + \exp(x_i^T \beta))$$
Logistic Regression in Python

```python
import numpy as np
from sklearn import datasets
dtrom sklearn.metrics import accuracy_score
drom sklearn.linear_model import LogisticRegression

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:50]
X_test = X[50:]
y_train = iris.target[:50]
y_test = iris.target[50:]

# model construction
model = LogisticRegression(C=10)

# model training
model.fit(X_train, y_train)

# model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
```
4. k-Nearest Neighbor

Q: how to classify the green test data?
Q: how to classify the green test data?

k-NN assigns an example to class $c$ if most of its $k$ nearest neighbors are from class $c$.

We could select $k$ nearest neighbors, or choose a distance threshold to find nearest neighbors.
Impact of Neighborhood Size

Q: how to classify the green test data?
k-NN assigns an example to class c if most of its k nearest neighbors are from class c.

Q: will k=3 and k=5 give the same result?
Impact of Neighborhood Size

Q: how to classify the green test data?
k-NN assigns an example to class c if most of its k nearest neighbors are from class c.

Q: will k=3 and k=5 give the same result?
Classification results depend on k.
Q: how to classify the green test data?
k-NN assigns an example to class c if most of its k nearest neighbors are from class c.

Q: will k=3 and k=5 give the same result?
Classification results depend on k.

Q: how does k affect model complexity?
Q: how to classify the green test data?
k-NN assigns an example to class \( c \) if most of its \( k \) nearest neighbors are from class \( c \).

Q: will \( k=3 \) and \( k=5 \) give the same result?
Classification results depend on \( k \).

Q: how does \( k \) affect model complexity?
Smaller \( k \) gives more complex model.
Impact of k on Model Complexity
k-NN in Python

```python
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:20]
X_test = X[20:]
y_train = iris.target[:20]
y_test = iris.target[20:]

# model construction
model = KNeighborsClassifier(n_neighbors=3)

# model training
model.fit(X_train, y_train)

# model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
```
5. Linear Support Vector Machine (LSVM)

Find a linear decision boundary to maximally separate instances from two classes.

\[ x^T \beta + \beta_0 = 0 \]

\[ M = \frac{1}{\|\beta\|} \]

LSVM typically works with binary classification, i.e., \( y = 0 \) or \( 1 \).
Learning LSVM

Find a linear decision boundary to maximally separate instances from two classes.

\[
\begin{align*}
    \max_{\beta, \beta_0, \|\beta\| = 1} M \\
    \text{s.t.} \quad y_i (x_i^T \beta + \beta_0) \geq M, \quad i = 1, \ldots, n
\end{align*}
\]
Soft-Margin LSVM

The boundary can tolerate misbehaved points (soft-margin).

\[
\begin{align*}
\max_{\beta, \beta_0, \|\beta\| = 1} & \quad M \\
\text{s.t.} & \quad y_i (x_i^T \beta + \beta_0) \geq M(1 - \epsilon_i), \quad i = 1, \ldots, n
\end{align*}
\]
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.svm import LinearSVC

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:-20]
X_test = X[-20:]
y_train = iris.target[:-20]
y_test = iris.target[-20:]

# Model construction
model = LinearSVC(C=10)

# Model training
model.fit(X_train, y_train)

# Model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
6. Decision Tree

Build a tree where each node filters some feature(s) for one-step classification.
Learning/Constructing a Decision Tree

Keep splitting nodes until leaf nodes are sufficiently “pure” or we run out of features.
Another Example

Build a tree to detect spam emails.

Features and Labels

X1 = color
X2 = “google”
X4 = “lottery”
R1, R3, R4 = spam
R2, R5 = hsm
Another Example

We want to have “pure” node.

Features and Labels

X1 = color
X2 = “google”
X4 = “lottery”
R1, R3, R4 = spam
R2, R5 = hsm
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn import tree

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:,-20]
X_test = X[-20:]
y_train = iris.target[:,-20]
y_test = iris.target[-20:]

# model construction
model = tree.DecisionTreeClassifier()

# model training
model.fit(X_train, y_train)

# model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
7. Neural Network

A network of neurons for classification.
A neuron is (typically) a non-linearized linear function of inputs.

Neuron

weights

activation functions

\[ f(x) = x \]
\[ f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \]
\[ f(x) = \frac{1}{1 + e^{-x}} \]
\[ f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \]
\[ f(x) = \tan^{-1}(x) \]
Backpropagation

BP is a popular algorithm to learn weights of a neural network. (~gradient descent)
First, we can write output of a neural network as a function of input features.

\[ g(x) = f_2 \left( \sum_j a_j \cdot f(w_{(j)}^T x) \right) \]
Backpropagation

Then, we take derivative of the function w.r.t. each weight

\[ \ell(\theta) = \sum_{i=1}^{n} (g(x_i) - y_i)^2 \]

\[ g(x) = f_2 \left( \sum_j a_j \cdot f(w_{(j)}^T x) \right) \]

\[ \frac{\partial g(x)}{\partial w_{(j)i}} \]

\[ \frac{\partial g(x)}{\partial a_t} \]

\[ \beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \]

\[ \alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}, \]
Neural Network in Python

```python
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:-20]
X_test = X[-20:]
y_train = iris.target[:-20]
y_test = iris.target[-20:]

# model construction
model = MLPClassifier(alpha=1e-3, hidden_layer_sizes=(2,3))

# model training
model.fit(X_train, y_train)

# model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
```
[+] Bayesian Network

Apply Bayesian network to infer $p(y = \text{cancer} \mid x = (\text{smoker, X-ray, Dyspnoea, Asia, } \ldots))$. 
Multi-Class Classification with Binary Classifiers

We can apply binary classifier by *one-versus-all* or *one-versus-one* strategies.
Machine Learning Prediction Models

Regression Model
- linear regression (least square, ridge regression, Lasso)

Classification Model
- naive Bayes, logistic regression, Gaussian discriminant analysis, k-nearest neighbor, linear support vector machine (LSVM), decision tree, neural network, (Bayesian network)
- multi-class classification with binary classifier

Advanced Model
- kernel, ensemble

Online Model
- SGD learner, HMM
1. Kernel Methods

Project data to a higher-dimensional space (implicitly) and do linear analysis there.
Demo
Implicit Mapping and Kernel Function

No need to know new feature representation; only need to know inner product of instances.

Kernel Function

$$\langle \phi(x), \phi(z) \rangle = \kappa(x, z) = 0.4$$

Example:

\[
\begin{bmatrix}
\@\$, \%\$,
\@\$
\end{bmatrix}
\begin{bmatrix}
gottery
gogog
loogle
\end{bmatrix} = 0.4
Representer Theorem

The optimal model (a.w. a proper objective function) is a linear combination of training data.

\[ L(\beta) = \sum_{i=1}^{n} (\phi(x_i)^T \beta - y_i)^2 + \lambda \beta^T \beta \quad \Rightarrow \quad \beta = \sum_{i=1}^{n} -\frac{1}{\lambda} (\phi(x_i)^T \beta - y_i) \cdot \phi(x_i) = \sum_{i=1}^{n} \alpha_i \phi(x_i) \]
Kernel Functions

Polynomial

$$K(q, q') = (1 + q \cdot q')^k$$

$$k(x, x') = ck_1(x, x')$$
$$k(x, x') = f(x)k_1(x, x')f(x')$$
$$k(x, x') = q(k_1(x, x'))$$
$$k(x, x') = \exp(k_1(x, x'))$$
$$k(x, x') = k_1(x, x') + k_2(x, x')$$
$$k(x, x') = k_1(x, x')k_2(x, x')$$
$$k(x, x') = k_3(\phi(x), \phi(x'))$$
$$k(x, x') = x^T A x'$$
$$k(x, x') = k_a(x_a, x'_a) + k_b(x_b, x'_b)$$
$$k(x, x') = k_a(x_a, x'_a)k_b(x_b, x'_b)$$

Sigmoid

$$K(q, q') = \tanh(aq \cdot q' + b)$$

Gaussian RBF

$$K(q, q') = \exp(-\|q - q'\|^2/\sigma^2)$$
Example Kernel Methods (in Python)

Kernel ridge regression

```python
import numpy as np
from sklearn import datasets
from sklearn import linear_model
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import mean_squared_error

# Data Preparation
diabetes = datasets.load_diabetes()
X = diabetes.data
X_train = X[:20]
X_test = X[20:]
y_train = diabetes.target[:20]
y_test = diabetes.target[20:]

# Model Construction (pick up of them)
# 2. ridge regression
model = linear_model.Ridge(alpha=1e-1)
model.fit(X_train, y_train)

# Model Training
model.fit(X_train, y_train)

# Model Testing/Assessment
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(mse)
```

Kernel LSVM (a.k.a. SVM)

```python
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:20]
X_test = X[20:]
y_train = iris.target[:20]
y_test = iris.target[20:]

# model construction
model = SVC(C=1, kernel='rbf', gamma='auto')
model.fit(X_train, y_train)

# model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
```

Kernel PCA

Kernel K-means

Gaussian Process
2. Ensemble Method

Ensemble several weak models to form a strong model.

Two popular ways to learn weak models

- bagging: bootstrapping + average
- boosting: weighting + weighted av
Bagging

Bagging has three steps

1. bootstrap training set to get subsets

2. learn one model from each subset

3. average model prediction

\[ f_{com}(x) := \frac{1}{m} \sum_{k=1}^{m} f_k(x) \]
An Example of Bagging: Random Forest

random forest = decision tree + bagging + feature bootstrapping
Boosting

Loop
- weight training instances
- train a model on the weighted sample

Get weighted average of model prediction

\[
f_{com}(x) = \sum_{k=1}^{m} \alpha_k f_k(x)
\]
An Example of Boosting: AdaBoost

Mis-classified instances receive higher weights; accurate models receive higher weights.
An Demo of AdaBoost
Example Ensemble Methods (in Python)

```python
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:-20]
X_test = X[-20:]
y_train = iris.target[:-20]
y_test = iris.target[-20:]

# Model construction
model = RandomForestClassifier(n_estimators=10)

# Model training
model.fit(X_train, y_train)

# Model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
```

```python
import numpy as np
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier

# Data Preparation
iris = datasets.load_iris()
X = iris.data
X_train = X[:-20]
X_test = X[-20:]
y_train = iris.target[:-20]
y_test = iris.target[-20:]

# Model construction
model = AdaBoostClassifier(n_estimators=10)

# Model training
model.fit(X_train, y_train)

# Model testing/evaluation
y_pred = model.predict(X_test)
TestErr = 1 - accuracy_score(y_test, y_pred)
print(TestErr)
```
Regression Model
- linear regression (least square, ridge regression, Lasso)

Classification Model
- naive Bayes, logistic regression, Gaussian discriminant analysis, k-nearest neighbor, linear support vector machine (LSVM), decision tree, neural network, (Bayesian network)
- multi-class classification with binary classifier

Advanced Model
- kernel, ensemble

Online Model
- SGD learner, HMM
How to learn a spam detection model from a sequence of emails?

- situation 1: have features and labels of received emails
- situation 2: only have labels of received emails
We can continually update model based on new emails, using stochastic gradient descent.

\[ L_n(f) = \sum_{t=1}^{n} \ell(y_t, f(x_t)) \]

\[ f = f - \eta \cdot \frac{\partial}{\partial f} L_n(f) = f - \eta \cdot \sum_{t=1}^{n} \frac{\partial}{\partial f} \ell(y_t, f(x_t)) \approx f - \eta \cdot \frac{\partial}{\partial f} \ell(y_t, f(x_t)) \]
Example: Perceptron

A single neuron that only updates itself when an instance is mis-classified.
A Demo of Perceptron
Q: how do we predict tomorrow’s weather based *solely* on historical weather records?
S2: Markov and Hidden Markov Models
Example Online Learners (in Python)

```python
1 import numpy as np
2 from sklearn import datasets
3 from sklearn.metrics import accuracy_score
4 from sklearn.linear_model import Perceptron

6 # Data Preparation
7 iris = datasets.load_iris()
8 X = iris.data
9 X_train = X[:-20]
10 X_test = X[-20:]
11 y_train = iris.target[:-20]
12 y_test = iris.target[-20:]
13
14 # model construction
15 model = Perceptron(alpha=0.01, tol=1)
16
17 # model training
18 model.fit(X_train, y_train)
19
20 # model testing/evaluation
21 y_pred = model.predict(X_test)
22 TestErr = 1 - accuracy_score(y_test, y_pred)
23 print(TestErr)
```