CS545: Distributed Computing Using Snowfall and Support Vector Machines

Chuck Anderson

Department of Computer Science
Colorado State University

Fall, 2009
Outline

Snowfall

Installation
Initialize a Cluster
Using the Snowfall Apply Functions
Libraries, Source Files, and R Objects
Example

Support Vector Machines
Large Margin Classifiers
Optimization
Kernels
Overlapping Distributions
Examples
Install Packages for Snowfall

- Check out the HighPerformanceComputing link at http://cran.r-project.org/web/views
Install Packages for Snowfall

- Check out the HighPerformanceComputing link at http://cran.r-project.org/web/views
- A good guide for using snowfall is Tutorial: Parallel Computing using R package snowfall
Install Packages for Snowfall

- Check out the HighPerformanceComputing link at http://cran.r-project.org/web/views
- A good guide for using snowfall is Tutorial: Parallel Computing using R package snowfall
Install Packages for Snowfall

- Check out the HighPerformanceComputing link at http://cran.r-project.org/web/views
- A good guide for using snowfall is Tutorial: Parallel Computing using R package snowfall
- You need snow and snowfall

```r
install.packages(c("snow","snowfall"))
```
Install Packages for Snowfall

- Check out the HighPerformanceComputing link at http://cran.r-project.org/web/views
- A good guide for using snowfall is Tutorial: Parallel Computing using R package snowfall
- You need snow and snowfall
  ```r
  install.packages(c("snow","snowfall"))
  ```
- We will set up a cluster that communicates with TCP/IP sockets, because this works on Linux and MS Windows without installing any additional software.
Initialize a Cluster

- To initialize a cluster

```r
sflInit(parallel = TRUE, cpus = 4, type = "SOCK")
```

Parallel may be set to FALSE to run on a single
Initialize a Cluster

- To initialize a cluster
  ```r
  sfInit(parallel = TRUE, cpus = 4, type = "SOCK")
  ```
  `parallel` may be set to FALSE to run on a single

- You may also specify which machines to use
  ```r
  sfInit(parallel = TRUE, type="SOCK",
         socketHosts=c("corn", "cucumber", "cucumber", "radish"))
  ```
  Without the `socketHosts` argument, you will be running on just your local host.
**Initialize a Cluster**

- To initialize a cluster
  ```
  sfInit ( parallel = TRUE, cpus = 4, type = "SOCK")
  ```
  `parallel` may be set to `FALSE` to run on a single

- You may also specify which machines to use
  ```
  sfInit ( parallel = TRUE, type="SOCK", 
          socketHosts=c( "corn", "cucumber", "cucumber", "radish" ))
  ```

  Without the `socketHosts` argument, you will be running on just your local host.

- Or, you may just call
  ```
  sfInit ()
  ```
  in your code and set the argument values in the R command line

  ```
  R --no-save --no-restore --args --parallel --cpus=4 \
  --type=SOCK
  ```

  The items that follow `--args` are parsed by `sfInit` using the R function `commandArgs`.
Which Hosts?

To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that
Which Hosts?

To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that

- consults a file of host names and maximum number of CPUs to use for each,
To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that

- consults a file of host names and maximum number of CPUs to use for each,
- uses the unix command `rup` to determine the current load on each host,
Which Hosts?

To simplify the creation of the `socketHost` host list, Andrew Sutton has written a clever R function, which I call `snowfallSelectHosts` that

- consults a file of host names and maximum number of CPUs to use for each,
- uses the unix command `rup` to determine the current load on each host,
- calculates the number of CPUs to use as the given maximum minus the current load,
Which Hosts?

To simplify the creation of the socketHost host list, Andrew Sutton has written a clever R function, which I call snowfallSelectHosts that

- consults a file of host names and maximum number of CPUs to use for each,
- uses the unix command `rup` to determine the current load on each host,
- calculates the number of CPUs to use as the given maximum minus the current load,
- duplicates the host name that many times, and returns the list of all host names.
Using `snowfallSelectHosts`

- Here is a text file of machine names and maximum numbers of CPUs:

```
brussels-sprout 5
cauliflower 5
horseradish 5
kelp 5
romanesco 5
```

These machines have 8 cores.
Using `snowfallSelectHosts`

- Here is a text file of machine names and maximum numbers of CPUs:
  - brussels-sprout 5
  - cauliflower 5
  - horseradish 5
  - kelp 5
  - romanesco 5

  These machines have 8 cores.

- If this file is names `machines`, then here is how I would use `snowfallSelectHosts`

```r
hosts <- snowfallSelectHosts("machines", localhost=TRUE, print=FALSE)
sfInit( parallel = TRUE, type = "SOCK", socketHosts = hosts)
snowfall 1.70 initialized : parallel execution on 17 CPUs.
```
Using `snowfallSelectHosts`

- You can also see the processing of each host.
  - Remember to stop the cluster we just created first.
Using `snowfallSelectHosts`

- You can also see the processing of each host.
  
  Remember to stop the cluster we just created first.

```r
sfStop()
hosts ← snowfallSelectHosts("machines", localhost = TRUE, print = TRUE)
```

which produces this output

```
Read 5 hosts from file "machines"
Using 0 of 5 slot(s) on brussels—sprout
Using 0 of 5 slot(s) on cauliflower
Using 0 of 5 slot(s) on horseradish
Using 3 of 5 slot(s) on kelp
Using 0 of 5 slot(s) on romanesco
Using 7 of 8 slot(s) on thoumire
for total of 10 slots
```

after which you continue with

```r
sfInit(parallel = TRUE, type = "SOCK", socketHosts = hosts)
snowfall 1.70 initialized: parallel execution on 10 CPUs.
```
Using new distributed apply functions

- Say we want to square each value of a list named `data`. Can use `sfLapply`.

```r
data <- 1:5
result <- sfLapply(data, function(x) {x * x})
print(result)
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9

[[4]]
[1] 16

[[5]]
[1] 25
```
Apply function with load balancing

- Can use the automatic load balancing provided by snowfall by using `sfClusterApplyLB`.

```r
data <- 1:5
result <- sfClusterApplyLB(data, function(x) {x * x})
print(result)
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9

[[4]]
[1] 16

[[5]]
[1] 25
```
Loading Libraries, Source Files, and Copying R Objects

Must also make sure each process loads the needed libraries and sources the needed R source files.

```r
sfLibrary (myneuralnet)
sfSource("/s/parsons/e/fac/anderson/tmp/nn.R")
```
Loading Libraries, Source Files, and Copying R Objects

- Must also make sure each process loads the needed libraries and sources the needed R source files.
  ```r
  sfLibrary("myneuralnet")
  sfSource("/s/parsons/e/fac/anderson/tmp/nn.R")
  ```

- Also must move all needed R objects, including data and functions, to each node.
  ```r
  sfExport("Xtrain","Xtest","Ttest","rmse","f","N","xmax")
  ```
## Distributed Training of Multiple Neural Networks

```r
### Function and Data to be approximated by neural network.

### function to be approximated

```R
f <- function(x) -1 + 0.05 * x + 0.4 * sin(x) + 0.1 * rnorm(length(x))
```

```r
N <- 40  
xmax <- 40
```

```r
Xtrain <- matrix(seq(0,xmax,length=N),N,1)  
Xtest <- Xtrain + xmax/N/2  
Ttest <- f(Xtest)
```

```r
### Helpful function

```R
rmse <- function(y,t) {
  sqrt(mean((y-t)^2))
}
```

```r
### Set up cluster

```R
library ( snowfall )  
source(" snowfallUtilities .R")  
## for selectHosts()a  
hosts <- selectHosts("machines",localhost=TRUE,print=FALSE)  
sfInit ( parallel = TRUE, cpus=40, type="SOCK", socketHosts=hosts)  
sfSource("/s/parsons/e/fac/anderson/lib/R/nn.R")  
sfExport("Xtrain","Xtest","Ttest","rmse","f","N","xmax")
```
```r
### function to run on each node. Argument is ignored
trainAndTest <- function(ijunk) {
    ## DON'T FORGET TO RENICE EACH PROCESS!!!!!!!!!
    system(paste("renice 19 -p", Sys.getpid()," >/dev/null"))
    hostname <- Sys.info()[$"nodename"]
    pid <- Sys.getpid()

    Ttrain <- f(Xtrain)
    nh <- 20
    nnet <- makeNN(Xtrain,Ttrain,nh,fPrec=0)
    ## return this network's prediction on the test input
    predictions <- useNN(nnet,Xtest)
    list ( predictions =predictions , info =info)
}

### Train this many networks
nreps <- 20
### Distribute the training of nreps networks over nodes using load balancing.
print(system.time( results <- sfClusterApplyLB( 1:nreps, trainAndTest)))
sfStop()  ## stop the cluster
```
pr <- par(mfcol=c(3,1))
### assemble predictions for each network
predictions <- NULL
for (i in 1:length(results))
  predictions <- cbind(predictions, results[[i]]$predictions)
### plot predictions for each network. Then plot the average prediction.
matplot(predictions, type="l", ylab="Predicted output")
plot(rowMeans(predictions), type="l", ylab="Average predicted output")
vsN <- NULL
for (i in 1:nreps) {
  vsN <- rbind(vsN, c(i, rmse(rowMeans(predictions[,1:i, drop=FALSE]), Ttest)))
}
plot(vsN[,1], vsN[,2], type="b", xlab="Number of Networks", ylab="RMSE")
print(vsN)
for (i in 1:length(results))
  cat(results[[i]]$info, " \n")
R Version:  R version 2.9.1 (2009-06-26)
snowfall 1.70 initialized: parallel execution on 32 CPUs.
  user  system  elapsed
  0.010   0.000  11.899

Stopping cluster

<table>
<thead>
<tr>
<th>nNodes</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>0.14635357</td>
</tr>
<tr>
<td>[2,]</td>
<td>0.12422137</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>[19,]</td>
<td>0.08956328</td>
</tr>
<tr>
<td>[20,]</td>
<td>0.08974663</td>
</tr>
</tbody>
</table>

Argument 1 ran on lang process 31507
Argument 2 ran on lang process 31515
...  
Argument 19 ran on horseradish process 3352
Argument 20 ran on horseradish process 3357
Results of Averaging Predictions of Multiple Neural Networks

![Graph of Predicted Output]

![Graph of Average Predicted Output]

![Graph of RMSE vs Number of Networks]
Outline

Snowfall

Installation
Initialize a Cluster
Using the Snowfall Apply Functions
Libraries, Source Files, and R Objects
Example

Support Vector Machines

Large Margin Classifiers
Optimization
Kernels
Overlapping Distributions
Examples
Large Margin Classifiers

- For a two-class classification problem, using target values of $-1$ and $1$, a sample $x_n$ is classified correctly by linear classifier if $t_n(w^T \phi(x_n) + b) > 0$.

$$\phi_1(x)$$

$$\phi_2(x)$$

$$y(x) = w^T \phi(x) + b = 0$$
Large Margin Classifiers

- For a two-class classification problem, using target values of $-1$ and $1$, a sample $x_n$ is classified correctly by linear classifier if $t_n(w^T \phi(x_n) + b) > 0$.

- Multiple lines work. Prefer the one for which the smallest perpendicular distance to a sample is maximized.
Perpendicular Distance

- What is perpendicular distance, $r$, from the line to a sample, $x$?
Perpendicular Distance

- What is perpendicular distance, \( r \), from the line to a sample, \( x \)?
Perpendicular Distance

What is perpendicular distance, $r$, from the line to a sample, $x$?

\[ y(x) = w^T x + b = 0 \]
Perpendicular Distance

What is perpendicular distance, \( r \), from the line to a sample, \( x \)?

\[
x = v + r \frac{w}{||w||}
\]

\[
w^T x + b = w^T v + b + w^T r \frac{w}{||w||}
\]

\[
y(x) = 0 + r \frac{w^T w}{||w||}
\]

\[
y(x) = r \frac{||w||^2}{||w||}
\]

\[
r = \frac{y(x)}{||w||}
\]
For a correctly classified sample, \( x_n, \ t_n y(x_n) > 0 \), so the distance of the sample to the boundary is \( \frac{t_n y(x_n)}{\|w\|} \).
For a correctly classified sample, \( x_n, t_n y(x_n) > 0 \), so the distance of the sample to the boundary is \( \frac{t_n y(x_n)}{\| w \|} \).

So, what we want is

\[
\arg\max_{w, b} \left( \min_n \frac{t_n y(x_n)}{\| w \|} \right)
\]

\[
= \arg\max_{w, b} \left( \frac{1}{\| w_n \|} \min_n t_n y(x_n) \right)
\]
For a correctly classified sample, \( x_n, t_n y(x_n) > 0 \), so the distance of the sample to the boundary is \( \frac{t_n y(x_n)}{||w||} \).

So, what we want is

\[
\text{argmax}_{w,b} \left( \min_n \frac{t_n y(x_n)}{||w||} \right)
\]

\[
= \text{argmax}_{w,b} \left( \frac{1}{||w||} \min_n t_n y(x_n) \right)
\]

This is difficult. Must simplify. Notice that

\[
\frac{t_n y(x_n)}{||w||} = \frac{t_n (w^T \phi(x_n) + b)}{||w||} = \frac{t_n (cw^T \phi(x_n) + cb)}{||cw||}
\]

for any \( c \).
For a correctly classified sample, \( x_n, t_n y(x_n) > 0 \), so the distance of the sample to the boundary is \( \frac{t_n y(x_n)}{\|w\|} \).

So, what we want is

\[
\arg\max_{w,b} \left( \min_n \frac{t_n y(x_n)}{\|w\|} \right) = \arg\max_{w,b} \left( \frac{1}{\|w_n\|} \min_n t_n y(x_n) \right)
\]

This is difficult. Must simplify. Notice that

\[
\frac{t_n y(x_n)}{\|w\|} = \frac{t_n (w^T \phi(x_n) + b)}{\|w\|} = \frac{t_n (cw^T \phi(x_n) + cb)}{\|cw\|}
\]

for any \( c \).

Let’s choose a \( c \) for which \( t(w^T \phi(x) + b) = 1 \) (once \( c \) is absorbed into \( w \) and \( b \)) for the sample \( x \) that is closest to the boundary. So \( t_n (w^T \phi(x_n) + b) \geq 1 \) for all \( n \).
Now our optimization problem is

$$\arg\max_{w,b} \left( \frac{1}{\|w_n\|} \min_n t_n y(x_n) \right)$$

$$= \arg\max_{w,b} \left( \frac{1}{\|w_n\|} \cdot 1 \right)$$

$$= \arg\min_{w,b} \frac{1}{2} \|w\|^2$$

with the constraint that $t_n(w^T \phi(x_n) + b) \geq 1$. 
Now our optimization problem is

\[
\arg\max_{w,b} \left( \frac{1}{\|w\|} \min_n t_n y(x_n) \right)
\]

\[
= \arg\max_{w,b} \left( \frac{1}{\|w\|} \cdot 1 \right)
\]

\[
= \arg\min_{w,b} \frac{1}{2} \|w\|^2
\]

with the constraint that \( t_n (w^T \phi(x_n) + b) \geq 1 \).

Can use algorithms designed for quadratic optimization subject to linear constraints to find optimum \( w \), but the following steps usually result in faster solutions.
Using Lagrange Multipliers

Use Lagrange multipliers, \( a = \{\alpha_1, \ldots, \alpha_N\} \), to include the constraints in the optimization problem.

\[
L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{n=1}^{N} \alpha_n (t_n(w^T \phi(x_n) + b) - 1)
\]

We want \( \arg\max_a \arg\min_w \min_b L(w, b, a) \)
Using Lagrange Multipliers

- Use Lagrange multipliers, \( \mathbf{a} = \{\alpha_1, \ldots, \alpha_N\} \), to include the constraints in the optimization problem.

\[
L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^{N} \alpha_n (t_n (\mathbf{w}^T \phi(\mathbf{x}_n) + b) - 1)
\]

We want \( \arg\max_{\mathbf{a}} \arg\min_{\mathbf{w}, b} L(\mathbf{w}, b, \mathbf{a}) \)

- How can we optimize this?
Gradients!

- First work on inner part (argmin)

\[
\frac{\partial L}{\partial w} = w - \sum_{n=1}^{N} \alpha_n t_n \phi(x_n) = 0
\]

\[
w = \sum_{n=1}^{N} \alpha_n t_n \phi(x_n)
\]

and

\[
\frac{\partial L}{\partial b} = - \sum_{n=1}^{N} \alpha_n t_n = 0
\]

\[
\sum_{n=1}^{N} \alpha_n t_n = 0
\]
Substituting these results into $L$ replaces $w$ and $b$ with expressions involving $\alpha_n$:

$$L(w, b, a) = \frac{1}{2} w^T w - \sum_{n=1}^{N} \alpha_n (t_n (w^T \phi(x_n) + b) - 1)$$

$$= \frac{1}{2} w^T w - \sum_{n=1}^{N} \alpha_n t_n (w^T \phi(x_n) + b) + \sum_{n=1}^{N} \alpha_n$$

$$= \frac{1}{2} w^T w - w^T \sum_{n=1}^{N} \alpha_n t_n \phi(x_n) - b \sum_{n=1}^{N} \alpha_n t_n + \sum_{n=1}^{N} \alpha_n$$

$$= \frac{1}{2} w^T w - w^T w + \sum_{n=1}^{N} \alpha_n$$

$$= -\frac{1}{2} w^T w + \sum_{n=1}^{N} \alpha_n$$

$$= \sum_{n=1}^{N} \alpha_n - \frac{1}{2} w^T w$$

$$= \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \phi(x_n)^T \phi(x_m)$$

such that $\alpha_n \geq 0$ and $\sum_{n=1}^{N} \alpha_n t_n = 0$. 

Chuck Anderson

Snowfall

Installation
Initialize a Cluster
Using the Snowfall
Apply Functions
Libraries, Source Files,
and R Objects
Example

Support Vector
Machines

Large Margin
Classifiers
Optimization
Kernels
Overlapping
Distributions
Examples
Can optimize (incorrectly) by simply climbing the gradient with respect to \( \mathbf{a} \) and force all \( \alpha_n \geq 0 \).

\[
\frac{\partial L(\mathbf{w}, b, \mathbf{a})}{\partial \alpha_k} = 1 - \sum_{n=1}^{N} \alpha_n t_n t_k \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_k)
\]
Can optimize (incorrectly) by simply climbing the gradient with respect to $a$ and force all $\alpha_n \geq 0$.

$$\frac{\partial L(w, b, a)}{\partial \alpha_k} = 1 - \sum_{n=1}^{N} \alpha_n t_n t_k \phi(x_n)^T \phi(x_k)$$

After climbing the gradient, can calculate

$$w = \sum_{n=1}^{N} \alpha_n t_n \phi(x_n),$$

and make predictions with

$$y(x) = w^T \phi(x) + b$$
Can optimize (incorrectly) by simply climbing the gradient with respect to \( a \) and force all \( \alpha_n \geq 0 \).

\[
\frac{\partial L(w, b, a)}{\partial \alpha_k} = 1 - \sum_{n=1}^{N} \alpha_n t_n t_k \phi(x_n)^T \phi(x_k)
\]

After climbing the gradient, can calculate

\[
w = \sum_{n=1}^{N} \alpha_n t_n \phi(x_n),\]

and make predictions with

\[
y(x) = w^T \phi(x) + b
\]

But, what is \( b \)? Can show (Appendix E) that solution \( a \) guarantees that

\[
\alpha_n \geq 0
\]

\[
t_n y(x_n) - 1 \geq 0
\]

\[
\alpha_n (t_n y(x_n) - 1) = 0
\]

So, for every sample, either \( \alpha_n = 0 \) or \( t_n y(x_n) = 1 \).
Can optimize (incorrectly) by simply climbing the gradient with respect to $a$ and force all $\alpha_n \geq 0$.

$$\frac{\partial L(w, b, a)}{\partial \alpha_k} = 1 - \sum_{n=1}^{N} \alpha_n t_n t_k \phi(x_n)^T \phi(x_k)$$

After climbing the gradient, can calculate $w = \sum_{n=1}^{N} \alpha_n t_n \phi(x_n)$, and make predictions with $y(x) = w^T \phi(x) + b$

But, what is $b$? Can show (Appendix E) that solution $a$ guarantees that

$$\alpha_n \geq 0$$

$$t_n y(x_n) - 1 \geq 0$$

$$\alpha_n (t_n y(x_n) - 1) = 0$$

So, for every sample, either $\alpha_n = 0$ or $t_n y(x_n) = 1$.

All samples for which $\alpha_n > 0$ are called support vectors.
So for support vector $x_n$,

$$t_n y(x_n) = 1$$

$$t_n(w^T \phi(x_n) + b) = 1$$

$$b = \frac{1}{t_n} - w^T \phi(x_n)$$
But, what about kernels?

- We found \( a = \{ \alpha_1, \ldots, \alpha_N \} \) that maximized \( L \). Many of the \( \alpha_i \)'s are zero. Let \( S \) be the set of sample indices for support vectors (the samples with \( \alpha_i > 0 \)). Rather than calculating the weight vector \( w \), we can leave the summation in place.

\[
\alpha_i = \frac{1}{t_n} - \sum_{s \in S} \alpha_s \phi(x_s)^T \phi(x_n)
\]
But, what about kernels?

- We found \( a = \{\alpha_1, \ldots, \alpha_N\} \) that maximized \( L \). Many of the \( \alpha_i \)'s are zero. Let \( S \) be the set of sample indices for support vectors (the samples with \( \alpha_i > 0 \)). Rather than calculating the weight vector \( w \), we can leave the summation in place.
- Since

\[
w = \sum_{s \in S} \alpha_s t_s \phi(x_s)
\]

and

\[
y(x) = w^T \phi(x) + b
\]

we can write

\[
y(x) = \sum_{s \in S} \alpha_s t_s \phi(x_s)^T \phi(x) + b
\]
But, what about kernels?

- We found \( \mathbf{a} = \{\alpha_1, \ldots, \alpha_N\} \) that maximized \( L \). Many of the \( \alpha_i \)'s are zero. Let \( S \) be the set of sample indices for support vectors (the samples with \( \alpha_i > 0 \)). Rather than calculating the weight vector \( \mathbf{w} \), we can leave the summation in place.
- Since
  
  \[
  \mathbf{w} = \sum_{s \in S} \alpha_s t_s \phi(x_s)
  \]

  and

  \[
  y(x) = \mathbf{w}^T \phi(x) + b
  \]

  we can write

  \[
  y(x) = \sum_{s \in S} \alpha_s t_s \phi(x_s)^T \phi(x) + b
  \]

- And, for \( b \), for support vector \( n \),

  \[
  b = \frac{1}{t_n} - \mathbf{w}^T \phi(x_n)
  \]

  \[
  = \frac{1}{t_n} - \sum_{s \in S} \alpha_s t_s \phi(x_s)^T \phi(x_n)
  \]
So, $\phi(x)$ only appears as a dot product with another $\phi(x)$. 

So, $\phi(x)$ only appears as a dot product with another $\phi(x)$. 

Key idea: Never have to explicitly calculate the feature vector $\phi(x)$. Why is this a good idea? If $\phi(x)$ is high dimensional, would be more efficient if we can calculate $\phi(x)^T \phi(v)$ in some way that doesn’t require calculating $\phi(x)$.

Let $k(x, v) = \phi(x)^T \phi(v)$. Can we just calculate $k(x, v)$?

Example (from Section 6.2). Let $x$ and $v$ be two-dimensional samples.

$k(x, v) = (x^T v)^2 = (x_1 v_1 + x_2 v_2)^2 = x_2 v_1 x_1 v_2 + 2 x_1 v_1 x_2 v_2 + x_2^2 v_2^2 = (x_2, \sqrt{2} x_1 x_2, x_2^2)(v_2, \sqrt{2} v_1 v_2, v_2^2)^T = \phi(x)^T \phi(v)$

What if $x$ and $v$ are 100-dimensional?
So, \( \phi(x) \) only appears as a dot product with another \( \phi(x) \).

Key idea: Never have to explicitly calculate the feature vector \( \phi(x) \). Why is this a good idea?
So, $\phi(x)$ only appears as a dot product with another $\phi(x)$.

Key idea: Never have to explicitly calculate the feature vector $\phi(x)$. Why is this a good idea?

If $\phi(x)$ is high dimensional, would be more efficient if we can calculate $\phi(x)^T \phi(v)$ in some way that doesn’t require calculating $\phi(x)$. 

\[ \begin{align*}
\text{Let } k(x, v) &= \phi(x)^T \phi(v) \\
\text{Example (from Section 6.2). Let } x \text{ and } v \text{ be two-dimensional samples.} \\
k(x, v) &= (x^T v)^2 = (x_1 v_1 + x_2 v_2)^2 = x_2 v_1^2 + 2x_1 v_1 x_2 v_2 + x_2^2 v_2^2 = (x_2 1, \sqrt{2} x_1 x_2, x_2 2)^T (v_2 1, \sqrt{2} v_1 v_2, v_2 2) \end{align*} \]
So, $\phi(x)$ only appears as a dot product with another $\phi(x)$.

Key idea: Never have to explicitly calculate the feature vector $\phi(x)$. Why is this a good idea?

If $\phi(x)$ is high dimensional, would be more efficient if we can calculate $\phi(x)^T \phi(v)$ in some way that doesn’t require calculating $\phi(x)$.

Let $k(x, v) = \phi(x)^T \phi(v)$. Can we just calculate $k(x, v)$?
So, \( \phi(x) \) only appears as a dot product with another \( \phi(x) \).

Key idea: Never have to explicitly calculate the feature vector \( \phi(x) \). Why is this a good idea?

If \( \phi(x) \) is high dimensional, would be more efficient if we can calculate \( \phi(x)^T \phi(v) \) in some way that doesn’t require calculating \( \phi(x) \).

Let \( k(x, v) = \phi(x)^T \phi(v) \). Can we just calculate \( k(x, v) \)?

Example (from Section 6.2). Let \( x \) and \( v \) be two-dimensional samples.

\[
k(x, v) = (x^T v)^2
= (x_1 v_1 + x_2 v_2)^2
= x_1^2 z_1^2 + 2x_1 v_1 x_2 v_2 + x_2^2 v_2^2
= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(v_1^2, \sqrt{2} v_1 v_2, v_2^2)^T
= \phi(x)^T \phi(v)
\]
So, $\phi(x)$ only appears as a dot product with another $\phi(x)$.

Key idea: Never have to explicitly calculate the feature vector $\phi(x)$. Why is this a good idea?

If $\phi(x)$ is high dimensional, would be more efficient if we can calculate $\phi(x)^T \phi(v)$ in some way that doesn't require calculating $\phi(x)$.

Let $k(x, v) = \phi(x)^T \phi(v)$. Can we just calculate $k(x, v)$?

Example (from Section 6.2). Let $x$ and $v$ be two-dimensional samples.

$$k(x, v) = (x^T v)^2$$

$$= (x_1 v_1 + x_2 v_2)^2$$

$$= x_1^2 z_1^2 + 2x_1 v_1 x_2 v_2 + x_2^2 v_2^2$$

$$= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(v_1^2, \sqrt{2} v_1 v_2, v_2^2)^T$$

$$= \phi(x)^T \phi(v)$$

What if $x$ and $v$ are 100-dimensional?
The matrix composed of all $k(x_n, x_m)$ is called the kernel matrix, or the Gram matrix. It must satisfy certain properties to be a valid kernel matrix, meaning one that can be formed by the dot product of feature vectors. (symmetric, positive semidefinite)
The matrix composed of all \( k(x_n, x_m) \) is called the kernel matrix, or the Gram matrix. It must satisfy certain properties to be a valid kernel matrix, meaning one that can be formed by the dot product of feature vectors. (symmetric, positive semidefinite)

Can combine kernel matrices to form new ones.
The matrix composed of all \( k(x_n, x_m) \) is called the kernel matrix, or the Gram matrix. It must satisfy certain properties to be a valid kernel matrix, meaning one that can be formed by the dot product of feature vectors. (symmetric, positive semidefinite)

- Can combine kernel matrices to form new ones.
- Another common example is the “Gaussian” kernel

\[
k(x, v) = e^{-||x-v||^2/2\sigma^2}
\]

The feature vector that corresponds to this kernel has infinite dimensionality!
• The matrix composed of all \( k(x_n, x_m) \) is called the kernel matrix, or the Gram matrix. It must satisfy certain properties to be a valid kernel matrix, meaning one that can be formed by the dot product of feature vectors. (symmetric, positive semidefinite)

• Can combine kernel matrices to form new ones.

• Another common example is the “Gaussian” kernel

\[
k(x, v) = e^{-||x-v||^2/2\sigma^2}
\]

The feature vector that corresponds to this kernel has infinite dimensionality!

• Can construct kernel matrices from samples with symbolic attributes. If \( A_1 \) and \( A_2 \) are two subsets of a given set, then the following is a valid kernel function.

\[
k(A_1, A_2) = 2|A_1 \cap A_2|
\]
Overlapping Class Distributions

- Above derivation assumed samples can be separated, so that

\[ t_{n y}(x_n) \geq 1 \]
Overlapping Class Distributions

- Above derivation assumed samples can be separated, so that

\[ t_n y(x_n) \geq 1 \]

- This assumption can be relaxed by allowing some "slack"

\[ t_n y(x_n) \geq 1 - \psi_n \]
Overlapping Class Distributions

- Above derivation assumed samples can be separated, so that
  \[ t_n y(x_n) \geq 1 \]

- This assumption can be relaxed by allowing some “slack”
  \[ t_n y(x_n) \geq 1 - \psi_n \]

- The goal is now to maximize the margin while softly penalizing samples that lie on the wrong side of the boundary. So, we want to minimize
  \[
  C \sum_{n=1}^{N} \psi_n + \frac{1}{2} ||w||^2
  \]
**Overlapping Class Distributions**

- Above derivation assumed samples can be separated, so that
  
  \[ t_n y(x_n) \geq 1 \]

- This assumption can be relaxed by allowing some "slack"
  
  \[ t_n y(x_n) \geq 1 - \psi_n \]

- The goal is now to maximize the margin while softly penalizing samples that lie on the wrong side of the boundary. So, we want to minimize
  
  \[ C \sum_{n=1}^{N} \psi_n + \frac{1}{2} ||w||^2 \]

- Doing the Lagrangian thing again and taking derivatives, we end up with the same optimization problem for the \( \alpha \)'s but different constraints
  
  \[ 0 \leq \alpha_n \leq C \]
  
  \[ \sum_{n=1}^{N} \alpha_n t_n = 0 \]
Example

- Here is an example of following the gradient to optimize $\alpha$'s (in svmGradient.R)
SVMs from the e1071 Package

“e1071: Misc Functions of the Department of Statistics (e1071), TU Wien”
SVMs from the e1071 Package

- “e1071: Misc Functions of the Department of Statistics (e1071), TU Wien”
- Support Vector Machines: The Interface to libsvm in package e1071, by David Meyer, Technische Universität at Wein, Austria
SVMs from the e1071 Package

- “e1071: Misc Functions of the Department of Statistics (e1071), TU Wien”
- Support Vector Machines: The Interface to libsvm in package e1071, by David Meyer, Technische Universität Wien, Austria
- libsvm: the award-winning (IJCNN 2001) C++-implementation by Chih-Chung Chang and Chih-Jen Lin
Example code from e1071 and our nnLogReg (in svmExample.R)

```r
svm.model <- svm(Type ~ ., data = trainset, cost = 100, gamma = 1)
svm.pred <- predict(svm.model, testset[, -10])

rpart.model <- rpart(Type ~ ., data = trainset)
rpart.pred <- predict(rpart.model, testset[, -10], type = "class")

nn.model <- makeNNLogReg(trainset[,1:9],trainset[,10,drop=FALSE],
                          nh=20,lambda=0.1,
                          fPrec=1e-6,xPrec=1e-8,nIter=10000)
nn.pred <- useNNLogReg(nn.model,testset[,1:9])
```