Like the time of compiler programming and Master programming List of things to be done on paper and cross out

General SAS: best resource: online search

31st June 2009

Any source/platform (data)→ management → analysis → presentation (any format)

Management: 1. Format 2. create variables (columns) 3. use operators to evaluate data values 4. use functions to create and recode data 5. subset data 6. perform conditional processing 7. merge a wide range of data sources 8. create, retrieve, and update database information.

Analysis: 1. produce tables, frequency counts, and cross-tabulation tables 2. create a variety of charts and plots 3. compute a variety of descriptive statistics, including the mean, sum, variance, standard deviation, etc. 4. compute correlations and other measures of association, multi-way cross-tabulations and inferential statistics.

output formats: 1. ML: HTML4 and XML 2. output that is formatted for a high-resolution printer, such PostScript, PDF, and PCL files, RTF 3. color graphs that you can make interactive using ActiveX controls or Java applets. 4. output these reports to a wide variety of locations and platforms

Result window: You can view, save, and print individual items of output. (Recall that the Results Viewer window is the where you actually view HTML output.)

SAS data set (table): descriptor (info related data values): organized as table of observation (rows) and variables (columns)

SAS library, where file such as SAS ***data sets & catalogs*** stored: Files in the directory; To access a library, you assign it a name (also known as a libref, or library reference).

Sashelp: library for SAS (Read only); Sasuser: to store your files, ***Permantnt***; work: ***temporary*** folder for not need to be saved

read other vendor's data directly using SAS/ACCESS.

```
* Create a SAS data set
named contest;
* Read the file Pumpkin.dat
using formatted input;
DATA contest;
INFILE"C:\CDBookSurvay\Pumpk
in.dat";
INPUT Name $16. Age 3. +1
Type $1. +1 Date MMDDYY10.
(Score1 Score2 Score3 Score4
Score5) (4.1);
RUN;
```

The +1 skips over one column.
By putting the variables and the informat in separate sets of parentheses, you only have to list the informat once.

P65

Format (right click column + attribute): a name; a *w* value, which specifies the width displaying; a period following the *w* value. Numeric SAS formats, such as the DOLLAR*w.d* format:*d* value, which is the number of decimal places t

table properties: Column (name, type, length, label)

An **informat** (input format) is the instruction that specifies how SAS reads raw data; the same as **Format** but start with $

SAS procedures help in: 1. manipulate data 2. store and retrieve information 3. perform statistical analysis 4. create reports.

SAS program two steps: a **DATA step** and a **PROC step**.

DATA steps typically create or modify SAS data sets, but they can also be used to produce custom-designed reports (e.g. put data in, compute value of new variable, check and correct data errors, produce new data by: subsetting, merging, and updating).

```
data clinic.admit2;
set clinic.admit;
run;
```

creates a new SAS data set ADMIT2 in the CLINIC library using the existing SAS data set ADMIT in the CLINIC library.

SAS procedure (step): data sets, such as listing, sorting, and summarizing data 1.print a report. 2. produce descriptive statistics. 3. create a tabular report 4. produce plots and charts.

```
procprintdata=clinic.admit2;
run;
```

prints the data in a data set. The DATA= option tells SAS what data to use for the procedure.

`SAS statement charactersties: 1. begins with a SAS keyword. 2. ends with a semicolon. A RUN statement tells SAS to process all the preceding lines of the step

```
DATA toads;
INFILE"C:\CDBookSurvay\ToadJum
p.dat";
INPUT ToadName $ Weight Jump1
Jump2 Jump3;
RUN; * Print the data to make
sure the file was read
correctly;
```

SAS Statement is free format: 1.can begin and end anywhere on a line 2.one statement can continue over several lines 3. several statements can be on a line. 4. SAS statements are not case sensitive. 5. Blanks or special characters separate the "words" in a SAS statement.

```
data test;/*10252012*/
       set
Ts102512.productsales;
run;
procprintdata=test;
run;
```

Defining one level name vs. multi level names

```
data original;
input x1-x5;
cards;
9 8 7 8 .
8 7 6 . 9
. . 9 7 6
run;
procprint;
Title'Original Data';
run;
data modified; set
original;
array zero x1-x5;
doover zero;
if zero=.then zero=0;
end;
procprint;
Title'Data modified
with arrray and do
loop';
run;
```

```
* Create a SAS data set
named distance;
* Convert miles to
kilometers;
DATA distance;
Miles = 26.22;
Kilometers = 1.61 *
Miles;
RUN;
* Print the results;
PROCPRINTDATA =
distance;
RUN;
```

```
*** Explicitly subscripted
array;
DATA ftoc2;
INPUT month $ f1-f7;
ARRAY f{7} f1-f7;
ARRAY c{7} c1-c7;
DO i=1to7;
      c{i}=( f{i}-32
)*5/9;
END;
FORMAT c1-c7 4.1;
CARDS;
aug     94 98 99 98 99 96
91 90 88 89
sept    93 92 87 87  89 90
91 92 82 80
;
run;
PROCPRINT;
title1'DATA; FTOC2';
title2'Explicit Array
Example';
RUN;
```

Import the data from excel/access/…

```
* Read internal data into
SAS data set uspresidents;
DATA uspresidents;
INPUT President $ Party $
Number;
DATALINES;
Adams F 2
Lincoln R 16
Grant R 18
Kennedy D 35
;
RUN;
```

```
* Read data from external
file into SAS data set;
DATA presidentt;
       INFILE"c:\CDBookSur
vay\president.dat";
       INPUT President $
Party $ Number;
RUN;
```

```
DATA sales;
INFILE"C:\CDBookSurvay\Oni
onRing.dat";
INPUT VisitingTeam $ 1-20
ConcessionSales 21-24
BleacherSales 25-28
OurHits 29-31 TheirHits
32-34 OurRuns 35-37
TheirRuns 38-40;
RUN;
```

Regular Expression	Char. Pattern matching perl	Multiple substitution in a string in one step	1. Match 2. Substitute 3. Split	Parse
Pattern matching via several steps	//$str =~ /this/i : i makes case insenstivie	$str =~ /-?23/; # i.e. 0 or 1 –'s	$str =~ /T*/; # matches 0 or more T's	

$str = “This is a string”;
$pattern = “ing”;
if ($str =~ /$pattern/){ print “match”;}
else { print “no match”;

+ means 1 or more

$str =~ /dog{4}y/; # matches 4 g’s

$str =~ /dog{1,5}y/; # matches 1,2, 3, 4, or 5 g’s

/(cat){3}/ # catcatcat
/cat{3}/ #cattt
\+ : to match +
\\: to match \

“.”: any char.
/.*/: any string even empty

\. : literal period escape it
“.” doesn’t match a newline
List of characters that need to be escaped: \ | / ( ) [ ] { } ^ $ * + ? .

my $str = “The dog”;
$str =~ /dog/; # matches
$str =~ /^dog/; # doesn’t work: “d” must be the first character
$str =~ /dog$/; # works: “g” is the last character

/"[^"]+"/# match quote, then everything that's not a quote, then a quote

$str = “There is a dog”;
$str =~ /The/ ; # matches
$str =~ /The\b/ ; # doesn’t match because the “e” isn’t at the end of the word (“\b”: bgn or end)

[135]: char. class match any 1, 3, or 5
[0-7]: matches digit 0 to 7; if hyphen first: [-0-9]

[^0-9]/ [^-0-9] matches any char which is not in class
[0-9\]]: match square bracket as well
[135]+: quantifier could also be used

Alternative: “|” : $str =~ /dog|cat|bird/;
# matches “dog” or “cat” or “bird”.

Unless want include other chars.:
\d = any digit = [0-9]
\s = whitespace (spaces, tabs, newlines) = [\t\n]
\w - word character = [a-zA-Z0-9_]
For **negation** capital: \D, \S, \W

Memory: saved in parentheses. The matching string is saved in scalar variables starting with $1.
$str = “The z number is z576890”;
$str =~ /is z(\d+)/;
print $1; # prints “567890”

Different variables are counted from left to right by the position of the opening parenthesis: /(the ((cat) (runs)))/ ;
captures: $1 = the cat runs; $2 = cat runs; $3 = cat; $4 = runs.

```
Multconcat.: my $string=join
"",$this, $that, $the, $other;
```

Memory inside regex: \1 isntead of $1:
/([AG]{3})CCGG\1/ matches AGGCCGGAGG
But it is match exactly although it is char class

When several match, only actual match captur.
$str = "My pet is a cat";
$str =~ /\b(cat|dog)\b/;
print $1; will print "cat".

Finding blank lines (space or tab):
/^\s*$/
matching letters only: /^A-Za-z$/ or /^[^\W\d_]$/

Substitute operator **works directly** on string

Greedy (+) vs. Lazy (?) matching:
$str = “The dogggg”;
$str =~ /The (dog+)/; #greedy
print $1; # prints dogggg
$str =~ /The (dog+?)/; # lazy
print $1; # prints “dog”
/ATG(.*?)TAG/ : everything b/w 1st ATG and 1st TAG
/ATG(.*)TAG/: everything b/w 1st ATG and last TAG

Warnining: \w does not match every words since they include: (‘), 1st, (-)

```
Concat: my $foo .= $bar;
```

Words can be found within words: "there goes the cat" =~ m/the/ matches the first word, not the fourth, but =~ m/\bthe\b/ matches the word "the" only.

```
# Remove all HTML except "p" tags
$html
=~s{<(?>/?)(?:[^pP]|[pP][^\s>/])[^>]
*>}{}g;
```

time of day: For example. 11:30. [01][0-9]:[0-5][0-9] won't work well, because it would allow such impossible times as 19:00 and 00:30. A more complicated construction works better: (1[012] | [1-9]) :[0-5][0-9]. That is, a 1 followed by 0, 1, or 2, OR any digit 1-9.

Concatenating string: my $s=$s.$b;or ="$s1$s2"

Things within a delimiter: e.g. <a href=http://www.bios.niu.edu> You want what's within the tags, everything between <a and >, so you try m/<a(.*?)>/. Need for lazy evaluation here, as there is probably more than one tag on the page. But still, this expression picks up the ending >, because it matches .*. So, use m/<a[^>]*>/ That is, zero or more characters that are not >. Usually you don’t really want .*

For nested html/xmal: non-word characters as delimiters: proceed with “m” inst. of “/” e.g. m<cat>, m@cat@ , m\cat\ (obfuscation)

Substitute one piece for another: $str =~ s/initial_pattern/substituted_chars/; only 1

BLAST scores (e-values) can be either decimals or exponents: 0.05 and 2e-40 and both typical values. To capture these numbers, use ([-e.\d]+). Also, sometimes BLAST scores start with e: e-35, for example. Perl doesn’t recognize this as a num., so you have to add a leading “1”: $score = "1" . $score if substr($score, 0, 1) eq "e";

the use of memory for capturing parts of the match:
$str = “I have 2 cats and 3 cars at home”;
@arr = split /a[rt]/, $str;
@arr has 3 elements: $arr[0] = “I have 2 c” , $arr[1] = “s and 3 c” , and $arr[2] = “s at home”.

$str = “A cat ia a nice pet”;
$str =~ s/cat/dog/;
print $str; # prints “A dog is a nice pet”

/s args: 1st: regular expression, met characters, assertions, alternatives, character classes and parentheses, capturing group within () also work

/s 2nd arg: NOT a regular expression, but one specific quoted string or variable ($1, $2, ..)

Substitution to remove characters: s/[^ACGT]//g

Substitute all: add a “g” to the end
$str = “A cat is a cat is a cat”;
$str =~ s/cat/dog/; # gives “A dog is a cat is a cat”
$str =~ s/cat/dog/g; # gives “A dog is a dog is a dog”

Making case insensitive: adding “i" to the end:
$str = “Cat”;
$str =~ s/cat/dog/; # no changes made; “cat” doesn’t match “Cat”
$str =~ s/cat/dog/i; # $str is now “dog”

Substitution and assignment:
($newstr = $oldstr) =~ s/cat/dog/;
# =~ > = keep

Translate tr///: pram: list of indv. char
$str = “ACCGTTAC”;
$str =~ tr/ACGT/TGCA/;
$str is now TGGCAATG

Converting all to uppercase:
$str =~ tr/a-z/A-Z/;

Count: $str = “AGCCTNNNCGTTANTA”;
$num = ($str=~ tr/ACGT//);
# returns the number of A, C, G and T, without counting the N’s.

find all instances of a pattern and give their positions.
while ($str =~ /ATG/g) {
my $position = pos $str;
my $start_position = $position - 4;
print “$start_position, “; }

position of the next character after the end of the matched string is found using “pos” followed by the name of the string variable being matched. (10/31/2012)

SAS regular expr:

Single step: PROC SQL and the %SYSFUNC macro command.
SAS(RX)
multiple step pattern: program editor Perl (PRX)

RXparse(f)	PRXparse(f)	Compile RX 4 pattern matching
RXMatch(f)	PRXMatch(f)	Search pttrn match & rtrn Pos. mtch fnd
Call RXSUBSTR(R)	Cal P..(r)	Rtrn POS & lngth of substr Match pttrn (RX includes scores
Call RXChange(r)	…P…(r) PRXchange (f)	Pttrn matching replacement
Call RXFree(r)	…P… ®	Free unneeded mem. Alloc 4 RX
The recommended best practice is to use the CALL routines in a data step;		
Use any expression in this list and search in SAS search		

```
proc sql;
   create table work.MarkTabTest as
   select S.*,
         prxparse("/\w*chips/") as re,
              ifc(
              prxmatch(calculated re,
              S.product),
              prxposn(calculated re, 0,
              S.product),
              " "
              ) as PRX_Return_String
   from sashelp.snacks as S;
   alter table work.MarkTabTest
       drop re;
quit;
```

Perl (PRX)	Description
CALL PRXPOSN(r)	Rtn start pos & length for capt. Buffer 2 substr
PRXPOSN(f)	Rtrn the val. 4 a capt. buff
PRXPAREN (f)	Rtrn the last brack. Match 4 which there is a match in pttrn
CALL PRXNEXT (r)	Rtrn the pos. & leng. Of a substr that matches a pttrn & iterate over multip match withn str
CALL PRXDEBUG (r)	Enables perl reg. exp. In a DATA step to send debug output to the SAS log

```
data_null_;
/* Use a pattern to replace all occurrences of cat, */
/* rat, or bat with the value TREE.                  */
length text $ 46;
   RegularExpressionId = prxparse('s/[crb]at/tree/');
text = 'The woods have a bat, cat, bat, and a rat!';
/* Use CALL PRXCHANGE to perform the search and replace.
*/
/* Because the argument times has a value of -1, the
*/
/* replacement is performed as many times as possible.
*/
```

```
data_null_;
ExpressionID = prxparse('/[crb]at/');
text = 'The woods have a bat, cat, and a rat!';
start = 1;
stop = length(text);/* Use PRXNEXT to find the first
instance of the pattern, */
/* then use DO WHILE to find all further instances */
/* PRXNEXT changes start param. so that searching */
/* begins again after the last match. */
call prxnext(ExpressionID, start, stop, text, position,
length);
dowhile (position >0);
```

```
call prxchange(RegularExpressionId, -1, text);
put text;
run;
```

```
        found = substr(text, position, length);
        put found= position= length=;
        if start > stop then position = 0;
        else
        call prxnext(ExpressionID, start, stop, text,
position, length);
end;
run;
```

RXparse : creating parsing function

Use this: To match this:
$a or $A : a-z A-Z
$c or $C : 0-9 a-z A-Z
$d or $D : 0-9
$l or $l : a-z A-Z (*only if first character in string*)
$l or $L : a-z
$u or $U : A-Z
$w or $W : whitespace

Communicate with other functions by passing value:

Rx = rxparse (.$character class.);

uge performance degradation will occur if this line is executed multiple times.

User defined characters: Rx = rxparse(.$ .A-Z.);

Character class complement:
Rx = rxparse(.^ .AEIOU.);

To preserved the original:
Call rxchange(rx, 1000, addr_1, newAddr_1);

To change (To keyword): in place
Rx = rxparse(.St. to Street.);
Call rxchange(rx, 1000, addr_1); # max number of times str to change

Use This: To Match This:
$f or $F Floating Point Number
$n or $N SAS Name
$p or $P Prefix (User Specified)
$q or $Q User specified String
$s or $S Suffix (User Specified)

Statistic: **RXSUBSTR**
addr_1 = ’.123 West St..:’
rx2 = rxparse(" St. to Street");
call rxsubstr(rx2,addr_1,position,length);
Will result in position = 10 and length = 3.

staterx = "'[$# ’North’ #9] | [$# ’South’ #4] |
[$# ’East’ #2] | [$# ’West’ #1] ";
rx = rxparse(staterx);
call rxsubstr(rx, addr_1, start, len, state);

Statistical characteristics of the data are examined using PROC UNIVARIATE. (normal tests normality hypothesis)

```
procunivariatedata=steam
plotnormal ;
var steamuse temp ;
title2'Univariate Descriptive
Statistics' ;
run;
```

scatter plots of the raw data in regression problems.

```
procplotdata=steam ;
title2'Scatterplot of Raw
Data' ;
plot steamuse*temp ;
run;
```

The PROC GPLOT : high-resolution graph of the raw data withthe reg line superimposed. Graph form is specified in the SYMBOLstatement, here: least squares reg line should be used to “interpolate" between data points and that raw data points should be indicated by plus signs.

```
procregdata=steam ;
title2'Least Squares
Analysis' ;
model steamuse = temp ;
run;
```

PROC MEANS or PROC CORR

Defining SAS library to work with data:

```
Libnamecdbktst"C:\";/*th
e place the files would
be put*/
run;
```

```
procgplotdata=steam ;
symboli=rl value=PLUS ;
plot steamuse*temp ;
title2'Observed Values and
Estimated Regression Line' ;
run ;
```

```
procsortdata=test ;
by id ;
run ; #patient diagnose
procmeansdata=test
noprint ;
by id ;
var dx1 dx2 dx3 ;
outputout=results max=;
run ;
procprintdata=results ;
run ;
```

```
PROCIMPORTDATAFILE =
"c:\CDBookSurvay\cdbook.
xls"DBMS=XLS
OUT=cdbkst.cdbk;
RUN; * Read an Excel
spreadsheet using PROC
IMPORT;
Proccontentsdata=
cdbktst.cdbk position;
Run; *content of the
file will be shown;
procmeansdata=cdbktst.cd
bk;
var age;
run; *mean of specific
data;
```

```
data cdbktst.cdbkusage;
*create new data set ;
set cdbktst.cdbk;
if BGHTCDST<4then offline=0;
else offline=1;
if BGHTCDON<4then online=0;
else online=1;
run;
```

```
procfreqdata=cdbktst.cdbkusag
e; * cross tab ;
tables offline*CDTRNS
online*CDTRNS/chisq;
run; *test of association b/w
CDTRNS and online offline ;
```

```
procfreqdata=
cdbktst.cdbkusage;
        tables offline online;
run; *create table of
frequency for new data;
```

```
procmeansdata=cdbktst.cdbkusa
ge;
        class CDTRNS;
run; *grouping the data based
on CDTRNS;
```

```
High resolution plot of
normal and exponential:
data randata;
drop i;
label normal_x = 'Normal
Random Variable'
exponential_x =
'Exponential Random
Variable';
do i = 1to100;
normal_x =
10*rannor(53124) + 50;
exponential_x =
ranexp(18746363);
output;
end;
run;
title'100 Obs Sampled
from a Normal
Distribution';
title2'Normal Q-Q Plot';
procunivariatedata=randat
a noprint;
qqplot normal_x /
normal(mu=est sigma=est);
insetmeanstd /
format=3.0header =
'Normal parameters'
position = se;
run;
```

```
proctimeseries
data=sashelp.air
out=series
outtrend=trend
outseason=season
print=seasons;
id date interval=qtr
accumulate=avg;
var air;
run;
```

```
procentropy data =
cdbktst.cdbk;
model BGHTCDST=RSKFRAUD
RSKSHIP RSKPERF RSKINFO
AVERRISK
HOURSONWEBVISTPRODINFONU
MPRUCH AVERUSE ASRTRNK
CONVRNK HASSRNK
ENJRNKINFORNK
        SERVRNKPRICRNK
        SEXAGEEDUC INCM;
run;
```

```
data melanoma ; *analysis of
unobserved component struct.
models;
input Incidences @@ ;
 year =
intnx('year','1jan1936'd,_n_-
1) ;
format year year4. ;
label Incidences = 'Age
Adjusted Incidences of
Melanoma per 100,000';
datalines ;
    0.9 0.8 0.8 1.3 1.4 1.2
1.7 1.8 1.6 1.5
    1.5 2.0 2.5 2.7 2.9 2.5
3.1 2.4 2.2 2.9
    2.5 2.6 3.2 3.8 4.2 3.9
3.7 3.3 3.7 3.9
    4.1 3.8 4.7 4.4 4.8 4.8
4.8     ;
run ;*src:
 http://support.sas.com/rnd/app
/examples/ets/melanoma/index.h
tm   /
```

```
procucm data = melanoma;
id year interval = year;
model Incidences ;
irregular ;
level ;
slope ;
cycle ;
run ;
odshtml ;
odsgraphicson ;
procucm data = melanoma;
id year interval = year;
model Incidences ;
irregular ;
 level variance=0 noest ;
 slope variance=0 noest ;
 cycle plot=smooth ;
estimate back=5 plot=(normal
acf);
forecast lead=10 back=5
plot=decomp;
run ;
odsgraphicsoff ;
odshtmlclose ;
```

```
procsortdata =
cdbktst.cdbkusage;
by BGHTCDST;
run;
```

```
procprintdata=cdbktst.cd
bkusage(obs = 7);
run; * only print 7
observation ;
```

```
data origdata; *Logit;
input ttime1 ttime2 ttime3
choice @@;
        datalines;
```

```
data enso(drop=pi);
*nonparametric model;
set enso;  pi = 4*atan(1);
  sin1=sin(2*pi*Month/12);
```

Conjoint sample

```
title'Nonmetric Conjoint
Analysis of Ranks';
procformat;
value BrandF
1 = 'Goodstone'
2 = 'Pirogi   '
3 = 'Machismo';
value PriceF
1 = '$69.99'
2 = '$74.99'
3 = '$79.99';
value LifeF
1 = '50,000'
2 = '60,000'
3 = '70,000';
value HazardF
1 = 'Yes'
2 = 'No ';
run;
data Tires;
input Brand Price Life Hazard
Rank;
format Brand BrandF9. Price
PriceF9. Life LifeF6. Hazard
HazardF3.;
datalines;
1 1 2 1  3
1 1 3 2  2
1 2 1 2 14
1 2 2 2 10
1 3 1 1 17
1 3 3 1 12
2 1 1 2  7
2 1 3 2  1
2 2 1 1  8
2 2 3 1  5
2 3 2 1 13
2 3 2 2 16
3 1 1 1  6
3 1 2 1  4
3 2 2 2 15
3 2 3 1  9
3 3 1 2 18
3 3 3 2 11;
proctransregmaxiter=50utiliti
esshort;
odsselect TestsNote
ConvergenceStatus
FitStatistics Utilities;
model monotone(Rank /
reflect) =
class(Brand Price Life Hazard
/ zero=sum);
outputireplacepredicted;
run;
procprintlabel;
var Rank TRank PRank Brand
Price Life Hazard;
label PRank = 'Predicted
Ranks';
run;
```

MCMC Logistic Bayesian:

```
data prior;
input _type_ $ Intercept
x;
datalines;
Var 25 25
Mean 0 0
;
run;
odsgraphicson;
title"Bayes with normal
prior";
procgenmoddescendingdata=
testmcmc;
model count/n =  x /
dist=binomial link=logit;
bayesseed=10231995nbi=100
0nmc=21000
coeffprior=normal(input=p
rior) diagnostics=all

statistics=(summaryinterv
al) plot=all;
run;
```

Clustering:

```
Procfastclusdata=cdbktst
.cdbkusage maxc=4
out=cdbktst.cdbcluster;
Var BGHTCDON BGHTBKST;
Run;
procplot;
plot
BGHTCDON*BGHTBKST=cluste
r;
run;
```

```
data endometrial;
*bayesian estimation;
input nv pi eh hg ;
nv2 = nv - 0.5; pi2 =
(pi-17.3797)/9.9978; eh2
= (eh-1.6616)/0.6621;
datalines;
0 13 1.64 0
0 16 2.26 0
0 8 3.14 0
0 34 2.68 0
0 20 1.28 0
0 5 2.31 0
0 17 1.80 0
0 10 1.68 0
1 11 1.01 1
1 21 0.98 1
0 5 0.35 1
1 19 1.02 1
0 33 0.85 1;
run;
procgenmoddescending;
model hg = nv2 pi2 eh2 /
dist=bin link=logit;
bayescoeffprior=normal
(var=1.0)
diagnostics=mcerrornmc=1
000000;
run;
procgenmoddescending;
model hg = nv2 pi2 eh2 /
dist=bin link=logit;
bayescoeffprior=normal
(var=100)
diagnostics=mcerrornmc=1
000000;
run; *src:
http://support.sas.com/d
ocumentation/cdl/en/stat
ug/63347/HTML/default/vi
ewer.htm#statug_genmod_s
ect007.htm ;
```

```
16.481 16.196 23.89  2 15.123
11.373 14.182 2
19.469  8.822 20.819 2 18.847
15.649 21.28  2
12.578 10.671 18.335 2 11.513
20.582 27.838 1
11.852 12.147 15.672 2 15.557
8.307 22.286 2 ;
run;
data newdata(keep=pid decision
mode ttime);
set origdata;
array tvec{3} ttime1 - ttime3;
*travel time (ttime1..ttime3);
retain pid 0;
pid + 1;
do i = 1to3; *extract whether
the choice is chosen;
mode = i; ttime = tvec{i};
decision = ( choice = i );
*not chosen:0, chosen:1;
output;
end;  *data format: panel
dataset where, in this case,
the variable pid indexes the
cross-section dimension
and the variable mode indexes
the time dimension.;
run;
procmdc data=newdata;
*conditional logit using
maximum likelihood;
model decision = ttime /
type=clogit nchoice=3
optmethod=qn covest=hess;
id pid; run;
```

```
DATA dads;     *father of
family;
INPUT famid name $ inc ;
CARDS;
2 Art  22000
1 Bill 30000
3 Paul 25000
;
RUN;
DATA faminc;  *income of
family;
INPUT famid faminc96 faminc97
faminc98 ;
CARDS;
3 75000 76000 77000
1 40000 40500 41000
2 45000 45400 45800
run;
PROCSORTDATA=dads OUT=dads2;
        *you must always sort
before merge
  BY famid;
RUN;
PROCSORTDATA=faminc
OUT=faminc2;
BY famid;
RUN;
DATA dadfam ;
MERGE dads2 faminc2;
BY famid;
RUN:
PROCPRINTDATA=dadfam;
RUN;
```

```
  cos1=cos(2*pi*Month/12);
run; *12 cycle time and
estimate residuals;
procregdata=enso;
model Pressure=sin1 cos1;
outputout=enso1
r=FilteredPressure;
run;
odsoutput
OutputStatistics=enso1Stats
FitSummary=enso1Summary;
procloessdata=enso1;
model
FilteredPressure=Month/smooth
=0.12
dfmethod=exact;run;
title1"Filtered ENSO Data";
symbol1color=black value=dot
i=noneh=3.5pct;
symbol2color=blue
interpol=join
value=nonewidth=2;
procgplotdata=enso1Stats;
format DepVar f2.0;
format Month  f3.0;
plot (DepVar
Pred)*Month/overlay
hminor = 0
vminor = 0
vaxis  = axis1
href = 4587129
frame;
axis1label = ( r=0a=90 )
order=(-6to6by2);
run;
```

Factor analysis: perception map: positioning & advertis.
Factor analysis (loading)

```
Data corrmatr (type=corr);
input M P C E H F;
Type = "CORR";
Cards;
0.37 0.62 0.54 0.32 0.284
0.37
0.62 1.00 0.51 0.38 0.351
0.43
0.54 0.51 1.00 0.36 0.336
0.405
0.32 0.38 0.36 1.00 0.686
0.73
0.284 0.351 0.336 0.686 1.00
0.7345
0.37 0.43 0.405 0.73 0.7345 1
run;
procfactormethod=prinit
rotate=v
corrmsascreeresidualspreplotp
lot;
var M P C E H F;
run;
```

R Programming Language	read.csv function: rds comma-delimited	>txt.csv <- read.csv("c:/temp/GLAccDesc.csv");	**R is case sensitive**
Tm package for text mining > library(tm) #loadlibrary > help (package=tm) #help of library	read.table function, which reads data that is tab-, comma-, or space-delimited [getReader() to see other]	"Suggests: filehash, proxy, Rgraphviz, Rmpi, RWeka, snow, Snowball, XML", Update other packages as well	up-to-date list of available datasources and readers available within the tm package: getSources()
cdbook.csv<- read.csv("c:/CDBookSurvay/cdbook.csv")	In order to analyze text it needs to be converted to **Corpus**. (readable by tm)	txt <- Corpus(DataframeSource(txt.csv))	**Note in csv: the file needs to be index since duplicate index is not allowed**

firefox.csv<-read.csv("c:/CDBookSurvay/firefoxanalyzed.txt")

firefox <- Corpus(DataframeSource(firefox.csv))

summary(firefox): summarize and inspect

inspect(firefox[1:5])

getTransformations(): available func. For preprocessing txt

firefox <- tm_map(firefox, tolower): convert to lower case

firefox <- tm_map(firefox, removeWords, stopwords("english"))#remove stop words
for (j in 1:length(txt)) txt[[j]] <- gsub("enterprise risk management", "erm",txt[[j]]) #replace wth abbr.

>txt <- tm_map(txt, removeNumbers)#remove numbers
>txt <- tm_map(txt, removePunctuation) #remove punctuations

tm_map(txt,tolower): convert to lowercase

for (j in 1:length(txt)) txt[[j]] <- gsub("/", " ",txt[[j]]): substituting "/" with" "

newstopwords <- c("and", "for", "the", "to", "in", "when", "then", "he", "she", "than") // use customized stop word in addition to normally available one
firefox <- tm_map(firefox, removeWords, newstopwords)

tm package stemming function: Remove words such as: "es", "ed" and "'s":
firefox <- tm_map(firefox, stemDocument)# stem words
firefox <- tm_map(firefox, stemCompletion, dictionary=firefoxcopy) #stem completion

dtm <- DocumentTermMatrix(firefox) : create document term Matrix

idx <- which(dimnames(myTdm)$Terms =="alexa")inspect(myTdm[idx+(0:5),1:10]) : show 5 terms in 10 document of dtm after term "alexa"

dtm3 <- removeSparseTerms(dtm, 0.94): removing sparse items of document term matrix

library(tm): load tm library

Remove numbers and punctuations:
firefox <- tm_map(firefox, removeNumbers)
firefox <- tm_map(firefox, removePunctuation)

Read dtm that is created in perl: dtm.csv<-read.csv("c:/Directory/dtm.csv")

Check the content of the dtm:
inspect(myTdm[0:10,1:10])

correlation is an indicator of how **closely related** two termsare**(similarity measure)**

finds all words with a correlation of at least:
findAssocs(dtm, "nice", 0.2):

Similaritymeasures can be applied**across rows** as well as **across columns** of a database. When applied **across rows** the similarityindicates **how similar two records** are. In the case of text mining, a similarity measure would **indicate how many words the two rows have in common**.

Chi- Squared: measures how closely related two categorical variables are,
Phi: measures the correlation b/w binary categorical variables

>library(proxy) # dissimilarity check
>dissimilarity(dtm3, method = "cosine")[1:10]

List of terms of dtm: rownames(myTdm)

Euclidian distance for dissimilarity (i, j: records, m:# variables) $d_{i,j} = (\sum_{k=1}^{m} (x_{i,k} - x_{j,k})^2)^{\frac{1}{2}}$

Cluster analysis: Maximize

```
require(vegan) ### some sample data
data(dune)# draw clauster
kclus <- kmeans(dune,centers= 4, # kmeans
iter.max=1000, nstart=10000)
dune_dist <- dist(dune) # distance matrix
# Multidimensional scaling
cmd <- cmdscale(dune_dist)
# plot MDS, with colors by groups from kmeans
groups <- levels(factor(kclus$cluster))
ordiplot(cmd, type = "n")
cols <- c("steelblue", "darkred", "darkgreen",
"pink")
for(i in seq_along(groups)){
  points(cmd[factor(kclus$cluster) == groups[i],
], col = cols[i], pch = 16)}
# add spider and hull
ordispider(cmd, factor(kclus$cluster), label =
TRUE)
ordihull(cmd, factor(kclus$cluster), lty =
"dotted")
```

data("crude")tdm <- TermDocumentMatrix(crude, control = list(removePunctuation = TRUE, removeNumbers = TRUE,stopwords = TRUE))

source("http://bioconductor.org/biocLite.R")
biocLite("Rgraphviz")

plot(dtm, terms = findFreqTerms(dtm, lowfreq = 1)[1:20], corThreshold = 0)# cluster draw

library(fpc) # draw clusters
plotcluster(dtm3, glKmeans$cluster)

Not removing 2 letter words
myTdm <- TermDocumentMatrix(firefox, control = list(wordLengths=c(1,Inf)))
myTdm

findFreqTerms(myTdm, lowfreq=3):
frequent words frequency not less than 3

```
x <- c(1,2,3,4,5,6)   # Create ordered collection (vector)
y <- x^2              # Square the elements of x
print(y)              # print (vector) y
 mean(y)              # Calculate average (arithmetic mean) of (vector)
y; result is scalar
> var(y)              # Calculate sample variance
> lm_1 <- lm(y ~ x)   # Fit a linear regression model "y = f(x)" or
"y = B0 + (B1 * x)"                  # store the results as lm_1
> print(lm_1)         # Print the model from the (linear model object)
lm_1
> summary(lm_1)       # Compute and print statistics for the fit
 # of the (linear model object) lm_1
> par(mfrow=c(2, 2))  # Request 2x2 plot layout
> plot(lm_1)          # Diagnostic plot of regression model
```

termFrequency <- rowSums(as.matrix(dtm))#highly used terms
termFrequency <- subset(termFrequency, termFrequency>=2)
library(ggplot2)
qplot(names(termFrequency), termFrequency, geom="bar") + coord_flip()

```
# plot of more frequent words [horizontal]
termFrequency <- rowSums(as.matrix(myTdm))
termFrequency <- subset(termFrequency,
termFrequency>=3)
library(ggplot2)
qplot(names(termFrequency), termFrequency,
geom="bar") + coord_flip()
# for vertical: barplot(termFrequency, las=2)
```

Find words that are highly associated with a word: findAssocs(myTdm, "love", 0.50)

Page 28 text mining hand book

Text mining

Text mining phases: 1. Preprocessing and integration of unstructured data, 2) statistical analysis of the preprocessed data to extract content from the text

unstructured text data is converted into structured data

Parsing: 1. Array of the words to be parsed 2. Search for space and record the position 3. Extract string from first position to position before space 4. Go to step 2 [split func. Of perl]

Simple analysis: Indicator 0/1 for existence of word e.g. "home owner": if the lower case of word is same then 1, else 0 (lc: lower case) => code

```
$target = "(homeowner)"; $i=0;
$flag=0; foreach $x (@words) {
if (lc($x) =~ /$target/) {
$flag=1; }}
```

Steps of preprocessing: 1.Parse the data. That is, extract the words from the data, typically **discarding** spaces and punctuation. 2. **Eliminate articles** and other words that convey **little or no info**. 3. **Replace** words that are **synonyms**, and **plural** and other **variants** of words with a singleterm. 4. Create the **structured data**, a table where each term in the text data becomes a **variable with a numeric** value for each record.

Parse: Extracting words:

```
@words =split (/[\s+]/,$Response3);
$len=@words;
for($i=0;$i<$len;$i++){
            print "$i 'th word is:
@words[$i]\n";
}
```

Removing the punctuations, and unwanted characters:

```
$Response3=~ s/[-.,?!"()'{}&;\/]//g;
```

Counting frequencies:

```
foreach $word (@words){
            ++$counts{lc($word)};
} #sort counts for keys
foreach $value(sort{$counts{$a}
cmp $counts{$b}} keys %counts){
            #print the word and the
count for the word
            print "$value
$counts{$value} \n";
}
```

Simple text statistics: 1. Length statistics of the word:
@countlen[length($word)] +=1

Produce matrix of term indicator:
1) create a **list of all words** in the data base (which will be referred to as the grand **dictionary**), 2) **check** each record (either **claim description or survey response**) for the presence of each word, 3) create an **indicator value of 1** if theword is present, otherwise a zero is recorded and 4) **print the results to a file** for further processing.

find a regular sentence structure that ends in a period ([^.]*\.)

Bag of words concept: order discarded and only **existence matter**(could be per sentence or per comment)

Create **term frequency DB**

Elimination of Stop words: the articles "the" and "a" [not add info.]by**substitu.**

```
#!perl –w: Create the matrix of term
# Program TermDocData.pl
# This program computes the term-document matrix
# a key part is to tabulate the indicator/count of every
```

```
#!perl –weliminate stop words while creating matrix of terms
# StopWords.pl
# This program eliminates stop words and computes the term-
document matrix
```

stemming:synonym & abbrev. handlin
important task and you can use normal DBs, yet, usually it needs to be **tailor made** based on context (**substitute**)

```
term - usually a word
# it may then be used to find groupings of words that create content
# This would be done in a separate program
# Usage: termdata.pl <datafile><outputfile>
$TheFile = "Top9.txt";
#$Outp1 = "OutInd1.txt";
# open input file with text data
open(MYDATA, $TheFile ) or die("Error: cannot open file");
# open first output file
open(OUTP1, ">OutInd1.txt") or die("Cannot open file for writing\n");
# open second output file
open(OUTP2, ">OutTerms.txt") or die("Cannot open file for writing\n");
# read in the file each line and create hash of words
# create grand dictionary of all words
# initialize line counter
$i=0;
# loop through data and convert to lower case and add to dictionary using hash
while (<MYDATA> ){
chomp($_);
$_ =~ lc($_);
s/[-.?!"()'{}&;]//g;
s/\s+/ /g;
@words = split(/ /);
foreach $word (@words) {
++$response[$i]{lc($word)}; # get freq of each word on line
++$granddict{lc($word)};}
++$i;}
# record no of lines in file
$nlines = $i-1;
print " no of lines is $nlines\n";
# print statitics to screen
for ($j=0; $j<= $nlines; ++$j ) {
print "$j ";
foreach $word (keys %{$response[$j]} )
{ print "$word, ${response[$j]{$word}},"; }
print "\n";}
# compute term-document matrix
# if term exists on record count frequency, else record gets a zero for the ter,
for $i (0..$nlines) {
foreach $word (keys %granddict) {
if (exists($response[$i]{$word}))
{++$ indicator[$i]{$word}; }
else{
$indicator[$i]{$word}=0;
}print OUTP1 "$indicator[$i]{$word},";
}print OUTP1 "\n";
}# print stats to file
foreach $word (keys %granddict) {
print OUTP2 "$word,$granddict{$word}\n";
}# close the files
close MYDATA;
close OUTP1;
close OUTP2;
```

```
# a key part is to tabulate the indicator/count of every term - usually a word
# it may then be used to find groupings of words that create content
# This would be done in a separate program
# Usage: termdata.pl <datafile><outputfile>
$TheFile = "Top2Iss.txt";
#$Outp1 = "OutInd1.txt";
open(MYDATA, $TheFile ) or die("Error: cannot open file");
open(OUTP1, ">OutInd1.txt") or die("Cannot open file for writing\n");
open(OUTP2, ">OutTerms.txt") or die("Cannot open file for writing\n");
# read in the file each line and create hash of words
# create grand dictionary of all words
# initialize line counter
$i=0;
while (<MYDATA> ){
chomp($_);
s/[-.?!"()'{}&;]//g;
s/^ //g;
s/,//g;
s/\d/ /g;
s/(\sof\s)/ /g;
s/(\sto\s)/ /g;
s/(\sthe\s)/ /g;
s/(\sand\s)/ /g;
s/(\sin\s)/ /g;
s/(The\s)/ /g;
s/(\sfor\s)/ /g
s/(\as\s)/ /g;
s/(A\s)/ /g;
s/(\sin\s)/ /g;
s/(\swith\s)/ /g;
s/(\san\s)/ /g;
s/(\swith\s)/ /g;
s/(\sare\s)/ /g;
s/(\sthey\s)/ /g;
s/(\sthan\s)/ /g;
s/(\sas\s)/ /g;
s/(\sby\s)/ /g;
s/\s+/ /g;
if (not /^$/) { #ignore empty lines
@words = split(/ /);
foreach $word (@words) {
++$response[$i]{lc($word)};
++$granddict{lc($word)};}
++$i;}
}$nlines = $i-1;
for $i (0..$nlines) {
foreach $word (keys %granddict) {
if (exists($response[$i]{$word})){
++$ indicator[$i]{$word}; }else{
$indicator[$i]{$word}=0;}
print OUTP1 "$indicator[$i]{$word},";}
print OUTP1 "\n";}
foreach $word (keys %granddict) {
print OUTP2 "$word,$granddict{$word}\n";}
# close the files
close MYDATA;
close OUTP1;
close OUTP2;
```

Similarity statistics of two texts:
cos(θ)=(A*B)/|A|*|B|, A, B: word freq.
Weighted frequency: term frequency-inverse document frequency (TF-IDF): shows importance of term, also adjusted for the number of records (or documents): down weight terms that exists everywhere
TF-IDF(i)= Frequency(i)*N/df(i),
df : #word frequency in all documents
N : # words in the record/document

Second step is: unsupervised learning
1. **no dependent variable** to fit amodel to. 2. use variables' values to **group like records** together

Perl (Quick and Nasty)

Scalar Variables

```
# End of line comments begin with a #
$a = 17;      # Scalar variables begin with a dollar symbol
              # The Perl assignment operator is =
              # Statements finish with a semicolon ;

$b = 0x11;    # Hexadecimal (17 in decimal)
$c = 021;     # Octal       (17 in decimal)
$d = 0b10001; # Binary      (17 in decimal)
$f = 3.142;   # Floating point

$a = $a + 1;  # Add 1 to variable $a
$a += 1;      # Add 1 to variable $a
$a++;         # Add 1 to variable $a

$b = $b * 10; # Multiply variable $b by 10;
$b *= 10;     # Multiply variable $b by 10;
# Other arithmetic operators include:
              #  **  Exponentiation
              #  %   Modulo division
              #  ++  Auto increment
              #  --  Auto decrement
              #  <   Numeric less than
              #  >   Numeric greater than
              #  ==  Numeric equality
              #  !=  Numeric inequality
              #  <=  Numeric less than or equal to
```

Scalar Variables Cont.

```
$a = 'Number of DFFs: '; # No interpolation with 'single quotes'
$b = "$a$c\n";           # Interpolation (variable substitution) with "double quotes"
# \n is the newline character
print $b;                # This makes "Number of DFFs: 17\n" appear on the standard output
print $a, $c, "\n";      # As does this line because print takes
#    a comma separated list of arguments to print
print "That's all\n";    # No commas means a list of one element

# String operators include:
                         #  lt  String less than
                         #  gt  String greater than
                         #  le  String less than or equal to
                         #  ge  String greater than or equal to
                         #  cmp String compare: Returns -1 0 1
print 'one' lt 'two';    # Prints 1
#    ASCII-betically 'o' is less than 't'
print 'buf4' lt 'buf3';
# Prints nothing (that is undef, numerically zero)
# Perl's undefined value is undef
#    ASCII-betically '4' is not less than '3'
```

Logic and Truth

```
0;      # Integer zero
  0.0;    # Decimal zero
  '0';    # String containing a single zero character
  '';     # Empty string
```

```
              #  >=  Numeric greater than or equal to
              #  <=> Numeric compare: Returns -1 0 1
```

```
  undef;  # Undefined
```

Logic and Truth

```
$a = 0; $b = 45;# More than one stmnt per line possible
print( $a and $b++ ); # prints 0         *
$a = 22;
print( $a and $b++ ); # prints 45        *
print $b;             # prints 46
                  # *  $b++ only evaluated when $a was
true
                      # Some logic operators take
shortcuts
                      # Other logical operators include
                      # or  Logical OR
                      # ||  Logical OR
                      # and Logical AND
                      # &&  Logical AND
                      # not Logical NOT
                      # !   Logical NOT
                      # |   Bitwise OR
                      # &   Bitwise AND
                      # ~   Bitwise NOT

print 6 & 5;          # prints 4, 0b0110 & 0b0101 =
0b0100
print 6 | 5;          # prints 7, 0b0110 | 0b0101 =
0b0111
print ! 0;            # prints 1
print ! 5;        # prints nothing (that is undef or
false)
print ~5;             # prints 4294967290, same as:
 # 0b11111111111111111111111111111010
```

Command Line Arguments

```
# This script is called process_netlist.pl
# Perl scripts often have the file extension .pl
$netlist_filename = $ARGV[0];
$report_filename  = $ARGV[1];
print "    Processing $netlist_filename\n";
print "    Writing report to $report_filename\n";
print "    ARGV contains '@ARGV'\n";
# Use it in this way:
#C:\perl process_netlist.pl chip_timesim.vhd report.txt
#Processing chip_timesim.vhd
# Writing report to report.txt
#ARGV contains 'chip_timesim.vhd report.txt'
# C:\
```

Arrays and Hashes

```
@components = ( 'X_LUT4', 'X_AND2', 'X_BUFGMUX', 'X_BUF_PP', 'X_FF' );
# or use qw''. Saves typing commas or quotes, gives the same result
# qw stands for Quoted Words
@components = qw'X_LUT4 X_AND2 X_BUFGMUX X_BUF_PP X_FF';
# or even put the data in columns, gives the same result again
@components = qw'
                    X_LUT4
                    X_AND2
                    X_BUFGMUX
                    X_BUF_PP
                    X_FF
                ';             # Easier to read this way
push( @components, 'X_MUX2' ); # Push another item onto the top
push( @components, 'X_ONE' );  # And one more
print $components[0];          # Prints element 0, that is, 'X_LUT4'
print $components[5];          # Prints element 5, that is, 'X_MUX2'
print "@components\n";         # Prints everything separated by spaces:
# X_LUT4 X_AND2 X_BUFGMUX X_BUF_PP X_FF X_MUX2 X_ONE
print  @components; # No double quotes,no spaces:
# X_LUT4X_AND2X_BUFGMUXX_BUF_PPX_FFX_MUX2X_ONE
```

```
while( @components ) {
#     ^^^^^^^^^^^^^^                     Array in scalar context
  $next_component = shift( @components );
  print "$next_component\n";
}
# Array variable @components is now empty
```

Arrays and Hashes

```
# Initialising several hash keys
%components = qw'
                  X_LUT4     0
                  X_AND2     0
                  X_BUFGMUX  0
                  X_BUF_PP   0
                  X_FF       0
                ';
#                 ^^^^^^^^^        keys
#                            ^   values
$components{'X_LUT4'} = 1; # Set key X_LUT4 to the value 1
$components{'X_LUT4'}++;   # Increment value associated with X_LUT4
print $components{'X_FF'}; # Print value associated with X_FF
@keys = keys %components;  # Get a list of hash keys
print "@keys\n";           # Print them - order is indeterminate
%components = ();          # Emptying the components hash
```

```
if( $ff_count == 1 )
#   ^^^^^^^^^^^^^^ Is this expression true or false?
{
  # Do this action if it is true
  print "There is 1 flip flop\n";
}
else
{  # Do this action if it is false
  print "There are $ff_count flip flops\n";
}
# More compact layout
if( $ff_count == 1 ) {
  print "There is 1 flip flop\n";
} else {
  print "There are $ff_count flip flops\n";
}
```

```
# Counting to one hundred
while( $count < 100 ) {
  $count++;
# Perl assumes $count == 0
the first time
  print "$count\n";
}
```

Read from screen & print

```
while( $line = <STDIN> ) {
$line_count++;
print "$line_count: $line";
}
# perl filter_netlist.pl <
chip_timesim.vhd> report.txt
```

```
foreach $course ( 'VHDL', 'SystemVerilog',
'SystemC', 'Perl', 'Tcl/Tk', 'PSL' ) {
  print "There is a $course Doulos training
course\n";
}
# $course is the loop variable.
# It takes the string value 'VHDL' for the
first loop
# and 'PSL' for the last loop.
# Get a list from an array variable
foreach $component ( @components ) {
  print "Component is $component\n";
}
```

cleaning strings safely: chomp($myvar);# changes $myvar

dropping the last character: chop($myvar); # changes $myvar

Files

```
open( FILE1, '>file1.txt' );
#             ^             > means open in write mode
print FILE1 "The first line to file1.txt\n";
print FILE1 "The final line to file1.txt\n";
close( FILE1 ); # Don't have to explicitly close a file
print STDOUT "This goes to the standard output\n";
print        "So does this\n";
#     ^^^^^^    STDOUT is a file handle that always
#               refers to the standard output.
# It is the default so doesn't have to be stated.
```

Files

```
open( FILE2, 'file2.txt' );  # Open in read mode - the default mode
$first_line = <FILE2>;#RD first line from file2.txt into $first_line
# Includes the newline character, \n.
while( $line = <FILE2> ) {
print $line;               # Read and print remaining lines from
file2.txt.
}                            # When every line has been read
<FILE2>returns undef.
$standard_input = <STDIN>;   # Read a line from the standard input.
# Can be the keyboard if run from the command line.
chomp( $standard_input );    # Remove the trailing newline character
```

```
$netlist_filename = $ARGV[0];# file names in the arg
$report_filename  = $ARGV[1]; #read from first file put
open( FILE_IN, $netlist_filename );
open( FILE_OUT, ">$report_filename" );
while( $line = <FILE_IN> ) {
  $line_count++;#number print in 2nd file
  print FILE_OUT "$line_count: $line";
}
# perl filter_netlist.pl chip_timesim.vhd report.txt
```

```
use English;
$string = "Novice to Expert in
a 3 day Perl course.\n";
if( $string =~ /\w+/ ) {
# \w+ matches one or more
alphanumeric characters in a row
  print "Matched: $MATCH\n";#
Matched:Novice}
```

```
use English;
$string = "Novice to Expert in a 3 day
Perl course.\n";
if( $string =~ /Perl\s+\w+/ ) {
# first part:           matches Perl
# second part: matches one or more white
space characters(including space, tab and
newline)
# 3rd part: matches one or more alphanumeric
chars.
  print "Matched: $MATCH\n";  #
Matched: Perl course
}
#  \w?    Zero or one letter, digit or
underscore
#  \w     One letter, digit or underscore
#  \w*    Zero or more letters, digits or
underscores
#  \w+  One or more letters, digits or
underscores
#  \W     One character but not a letter,
digit or underscore

#  \s White space character, space, tab or
newline
#  \S One char but not a space, tab or
newline
```

Pattern matching (String)

```
$string = "Novice to Expert in a 3 day Perl course.\n";
print $string;
if( $string =~ m/Expert/ ) {
# A successful match returns 1 so this statement is executed
  print "This string contains the substring
'Expert'\n";
}
# m stands for match
# Forward slashes are used to /delimit/ regular expressions.
# =~ tells the m operator which string to search.
# The m is optional when // are used.
```

Grouping:

```
open( VHDL_FILE, 'chip_timesim.vhd' );
```

Quick intro contd.:

● quoted strings:

➔*"$xyz and other stuff \t \n"*: like C printf, variables are substituted

➔*'$xyz and other stuff \t \n'*: literal printing

● operators: numeric vs string

➔numbers: ==, >, <, >=, <=, !=; <=>: *returns -1, 0, or 1*

➔strings: *eq, gt, lt, ge, le, ne*

● lexicographic comparison: 300 <= 40 is false, 300 le 40 is true

● lists and arrays

➔list syntax: *("abc", "def", "etc"); qw(abc def etc);*

```
while( $line = <VHDL_FILE> ) {
  if( $line =~ /\w+\s*:\s*(X_\w+)/ ) {
#1st element:  Instance label
#2nd element: Zero or more white space characters
#3rd element : “:”
#4th element: Zero or more white space characters
#5th element: Group containing a word beginning with X_
#  (copied into $1)
    print "Found instantiation of $1\n";
  }
}
```

Sort array and use function

```
@arr=(3,4,5,2,1);
print "main array is:@arr";
@sort=sort @arr;
print "sorted array is:@sort";
sub myfunc {print"@_";return reverse @_;}
@rev=myfunc(@arr);
print "main arryas:@arr";
print "reverse array:@rev";
```

➔array: @myarr = (“abc”, “def”, “etc”);
➔accessing: $myarr[3];
@myarr[0..@myarr]; #ranges, returns array
➔array mode assignment/access:
@newarr = @myarr;
@newarr = (@myarr, “append this”);
print @myarr; #array mode: prints array entries, concatenated
print @myarr . “\n”; # string mode: length of @myarr
($a, $b, $c) = @myarr;
➔scalar mode assignment/access:
$a = @myarr; # length of @myarr
print $#myarr; # idx of last element of @myarr
➔reading in multiple lines: @manylines = <STDIN>; # end with EOF=^D

- command line argument array: @ARGV

NetList Filtering: [Getting name com components: hashtable + grouping+ counting]

```
# Pulling it all together
# Everything in this script is described
above
$netlist_filename = $ARGV[0];
open( VHDL_FILE, $netlist_filename );
while( $line = <VHDL_FILE> ) {
  if( $line =~ /\w+\s*:\s*(X_\w+)/ ) {
    $component_hash{$1}++;
  }
}
@name_array = keys %component_hash;
foreach $component_name ( @name_array
) {
  print "$component_name:
$component_hash{$component_name}\n";
}
```

useful functions for arrays

- push, pop, reverse
- shift; unshift
- sort; sort {$a <=> $b} @myarr;
- if/then/else: if {...} elsif {...} else {...}
- loops
- for($i=0; $i<10; $i++) {...};
- foreach $i (@myarr) {...};
- implicit scalar variable $_: foreach (@myarr) {print; # $_};
- perl references: foreach (@myarr) {$_ *= 2;}; #changes @myarr
- while ($i<100) {...};
- until($i==100) {...};
- do {...} while ($i<100); do {...} until ($i==100);
- functions
- sub myfunc {print “@_”; return reverse @_;}
- ($a, $b, $c) = myfunc(qw(a b c d e)); # like Matlab

- file existence tests (a la bash's [-X filename]):
- -e, -f, -d, -l, -r, -w, -x
- hashes (associative arrays): %myhash
- simple assignment: $myhash{“abc”} = “def”; #sort of like Matlab cell
- list of keys: @mykeys = keys(%myhash); if (keys(%myhash)>5) {...};
- list of values: @myvals = values(%myhash);
- foreach $myval (keys(%myhash)) { ... };
- while (each($mykey,$myval)) { ... };
- delete $myhash{$mykey};
- hash to array conversion: @myarr = %myhash;
- array to hash conversion: %myhash = @myarr; %myhash = (1,2,3,4);
- scalar access of hashes: if (%myhash) {...};
- hash slices:
- @myhash{@mykeys} = @myvals;
- @existinghash{keys(%myhash)} = values{%myhash};
- print “@myhash{@mykeys} @myhash $myhash”;

```
Remove HTML everything except <\p>:
s{
<# opening angled bracket
(?>/?)# ratchet past optional /
(?:
[^pP]# non-p tag
|# ...or...
[pP][^\s>/]# longer tag that begins with p
(e.g., <pre>)
)
[^>]*# everything until closing angled
bracket
># closing angled bracket
}{}gx;# replace with nothing, globally
```

- string matching, substitution, splitting
- if ($mystr =~ /$someRE/) { ... }; $mystr =~ s/$myRE/$otherRE/;
- $mystr = "This is a istring"; $oof = "^This(.*)(.*)\\1(.*)\$";
- if ($mystr =~ m@/withslashes/@) { ... };
- $mystr =~ s/$oof/$1,$2,$3/;
- Perl regular expressions:
- spaces, “nonspace”, digits: \s, \S, \w, \W, \d, \D
- any char, multiple occurrences: ., *, +
- word boundaries: \b, \B
- “greediness”: default is max; for min, follow by ?: *?, +?, etc.
- m/(...).*(...)/; print “$1 $2”;
- OR: |
- @myarr = split(/\s+/, $mystring);
- $mystring = join(',', @myarr);
- file opening/closing/access
- open(FH,”filename”); open(FH,”<”, $filename); open(FH,”>”, ...);
- close(FH);
- die: open(FILEHANDLE,”filename”) || die “open failed: $!”;
- opendir, unlink, rename, chmod, chdir, etc. (man perlfunc)
- opening/reading all cmdline args as files:
- while (<>) {echo $_;} # no args? read stdin

Perl references

- all references are scalars
- $myref = \@myarr; $myref = \$myscalar; $myref = \%myhash;
- shortcuts for references to arrays and hashes
- $myref = [“a”, “b”, “c”]; # same as @myarr = qw(a b c); $myref=\@myarr;
- (anonymous array/hash created: a bit like using malloc/new)
- $myref = { “key1” => “val1”, “key2” => “val2” };
- $reftoemptyarr = []; $reftoemptyhash = {};
- dereferencing: enclose reference within {}
- $myref = \@myarr; @newarr = @{$myref}; # same as @newarr=@myarr;
- $fourthmem = ${$myref}[3]; # same as $fourthmem = $myarr[3]
- $myref=\%oldhash; %newhash= %{$myref}; # same as %newhash = %oldhash;
- ${$myref}{“key”} = “val”;
- copies of references are still references (think C pointers)
- $newref = $myref; # like C pointers, not C references!
- C pointer like syntax
- $myref->{“key”} = “val”; # equivalent to ${$myref}{“key”} = “val”;
- $myref->[2] = 5.6; # equiv to ${$myref}[2] = 5.6;
- multidimensional arrays in Perl
- @my2darr = ([1, 2, 3], [4, 5, 6], [7, 8, 9]);
- @my3darr = ([[1,2], [3,4]], [[5,6],[7,8]]);
- $my2darr[1]->[2] = “was6”;
- $my3darr[1]->[1]->[0] = “was7”;
- more shortcut notation
- $my3darr[1][1][0] = “was7”; # drop multiple ->: same as $my3darr[1]->[1]->[0] !
- @my3darr[1][1]; # same as @my3darr[1]->[1], == (7,8)

- string matching, substitution, splitting
- if ($mystr =~ /$someRE/) { ... }; $mystr =~ s/$myRE/$otherRE/;
- $mystr = "This is a istring"; $oof = "^This(.*)(.*)\\1(.*)\$";
- if ($mystr =~ m@/withslashes/@) { ... };
- $mystr =~ s/$oof/$1,$2,$3/;
- Perl regular expressions:
- spaces, “nonspace”, digits: \s, \S, \w, \W, \d, \D
- any char, multiple occurrences: ., *, +
- word boundaries: \b, \B
- “greediness”: default is max; for min, follow by ?: *?, +?, etc.
- m/(...).*(...)/; print “$1 $2”;
- OR: |
- @myarr = split(/\s+/, $mystring);
- $mystring = join(',', @myarr);
- file opening/closing/access
- open(FH,”filename”); open(FH,”<”, $filename); open(FH,”>”, ...);
- close(FH);
- die: open(FILEHANDLE,”filename”) || die “open failed: $!”;
- opendir, unlink, rename, chmod, chdir, etc. (man perlfunc)
- opening/reading all cmdline args as files:
- while (<>) {echo $_;} # no args? read stdin

- shell commands and system interaction:
- $stdoutput = `date`;
- $retval = system(“date”); # $? is returned
- environmental variables: %ENV
- eval
- $a='$b'; $b='$c'; $c=”oof”; eval “\$a=$a”;

Latex						

\documentclass{article}
\begin{document}
A minimal \LaTeX\ document.
\end{document}
	Windows
	– proTEXt ! LATEX processor
	– TeXnicCenter ! editor
	Landscape: \special{landscape}
		\alpha
		\psi
		\omega
			xy \pm ±
			\times ×
			\approx
			\int_0^\infty
			\int{\int}
				x^y
				x_y
				x_y^z
					$\frac{\partial u}{\partial x}$
					\begin{figure}
					\includegraphics{graph}
					\end{figure}
						\begin{document}
						\end{document}

$ \alpha=\frac{\beta}{\gamma}	\# \$ \% \& \~ _ \^ \\ \{ \}		

\title{Social network in search}
\author{Meisam Hejazinia \\ Brian Ratchford \\ Ernan Haruvey}
\date{5 December 2012}
\maketitle
	There has to be at least two new line to separate the paragraphs.
	To double space:
	\renewcommand{\baselinestretch}{2}
		\begin{tabular}{|l|r|c|}
		\hline
		Person & Money Owing & Silly Com \\ \hline
		Mr. C & \$1943.12 & pay him agn, Sam \\ \hline
		Mr. P & \$55.55 & what robbery? \\ \hline
		Mr. Sc & \$666.00 & the golden rule \\ \hline
		Mr. Ca& \$300.51 & bad accountants \\ \hline
		\end{tabular}
		\begin{figure}
			Document Wide Stuff:
			\documentstyle[options]{style}
			style: article report book slides
			options: 11pt 12pt twoside twocolumn titlepage leqn openbib fleqn
			\pagestyle{style}

Title Page Stuff:
\maketitle
\begin{titlepage} ... \end{titlepage}
\begin{abstract} ... \end{abstract}
	Bibliography and Citation
	There is (well, it'll be here RSN) some more information here in this document that gives an example of the		

Cross reference:
\label{key}
assign current counter value to key
\ref{key}
print value assigned to key

Input from Different Files:
\input{file}
read the file
\include{file}
read the file unless not in
\includeonly{}
\includeonly{filelist}
exclude any file not in filelist

Lists:
\begin{itemize} ... \end{itemize}
a 'bulleted' list
\begin{enumerate} ...
\end{enumerate}: a numbered list
\begin{description} ...
\end{description}:a list of labeled items

```
\begin{thebibliography}{9}          %
9 = maximum expected references!
\bibitem{Lam} Lamport, Leslie.
\LaTeX : A Document Preparation
System.  \\
Copyright \copyright 1986, Addison-
Wesley Publ.Co.,Inc.
\bibitem{Sch} Schl\oe
ff\d{o}nffl\t{oo}\ae g\"{e}n,
\L\"{a}rs.  Silly Typography.        \\
{\em Journal of Linguistic Horseplay
19D} (1977), 23-37.
\end{thebibliography}
\end{document}
```

To include images in the document this package should be used.
\usepackage{graphicx}

For probability use: \sim

bibtex stuff.
\bibliography{...}
\begin{thebibliography} {label} ...
\end{...}
make bibliography; lable is the widest entry label
\bibitem[label]{key}
begin bibliography entry for citation key
with label as its label
\cite[note]{keys}
cite reference(s) keys with added note

\begin{quote} ... \end{quote}
short displayed quotation
\begin{quotation} ... \end{quotation}
long displayed quotation
\begin{flushleft} ... \end{flushleft}
left flush lines, separated by \\
\begin{center} ... \end{center}
centered lines, separated by \\
\begin{flushright} ... \end{flushright}
right flush lines, separated by \\
\begin{verse} ... \end{verse}
\\ between lines, blank line between stanzas
\begin{verbatim} ... \end{verbatim}
Fixedlength, typewriter face exactly as formatted use any characters you like!

Math:
`$ ... $` or \(... \) :Intext formulas
\[.. \] : displayed formulas
\begin{equation} ... \end{equation}
a numbered equation
\begin{eqnarray} ... \end{eqnarray}
numberedequation, like 3 column array environm.
\nonumber omits one equation number, eqnarray* omits all
_{ ... }
subscript. NB: don't need the braces for one character
^{ ... }
superscript. NB: don't need the braces for one character
'   :prime
\frac{n}{d} print the numerator over the denominator
\sqrt[n]{arg}   the nth root of the argument arg
ellipsis \ldots ... \cdots ... \vdots ...
Greek letters \alpha ... \omega and \Alpha ... \Omega
delimiters \left or \right followed by delimiters
\overline{expression}
print a rule over the expression
space thin \. medium \: thick \; negative thin \!

```
\begin{em}
   A long segment of
\end{em}
```

style: plain empty headings
myheadings
\pagenumbering{style}
style: arabic roman alph Roman
Alph

```
`\begin{eqnarray*}
%  "*" = no line numbering
  \sum_{n=1}^k \frac1n
& \approx &\ln k + \gamma  \\
& = & (\ln 10)(\log_{10}k) +
\gamma \\& \approx
&2.3026\log_{10}k + 0.57772
\end{eqnarray*}
```

```
\section{Ordinary Text}       %
Produces section heading.  Lower-
level sections
% \subsubsection commands;
numbering is automatic!
```

```
\subsection{Spacing in the source
text}
```

```
\indent  {\bf Bold face type,}
\indent  {\tt typewriter style type,}
\indent  {\sf sans-serif type,}
\indent  {\sl slanted type,}    \\
\indent  {\sc all caps type.}   \\
```

```
\begin{equation}  a^{p} + b^p
\neq c^{p} ~~~\mbox{for } p>2
~~ \mbox{(see proof in margin)}
\label{eq:fermat}
\end{equation}
$$ \lim_{n \rightarrow
\infty}x_{n} \geq \pi $$
$$ \forall x \in {\cal O} ~~\exists
\delta ~~~\mbox{such that}~~~
|y-x|<\delta ~\Rightarrow ~y \in
{\cal O} $$
\vspace{4mm}$$
\Psi' = \frac{d}{d \phi} \left(
\begin{array}{c}
 \phi_{2}  \\  \phi_{3}  \\  1 -
\phi_{2} - \phi_{1}^{2}/2
 \end{array} \right)
 ~~~~~~~~~~~~~~
 \Theta =
\left(\begin{array}{ccc} 0
- \theta_{1} \psi_{1} - \psi_{2}
&0& \psi_3    \\  -\phi_{1}
$$ \vspace{4mm}
```

#PROGRAM MATCHLINE.TXT TO SEARCH FOR THE PHRASE THAT MOST CLOSELY MATCHES A PHRASE.

It finds the record most similar tothe input phrase “Credibility of the CAS”, using the cosine, measure, though other similaritymeasures could be used instead
1. Read the database.
2. Create a hash of all words on each record.
3. Create a hash of all words in the database.
4. Compute the TF-IDF statistic for each term on each record of the database.
5 Read the search string.
6. Compute the TF-IDF for the search string.
7. Compute the cosine correlation between the TF-IDF of the search string and each record inthe database.
8. Determine which record is the closest match and print it out.

```
if (exists $tf[$i]{$word} ) {
$tf_val = $tf[$i]{$word};
}else {$tf_val = 0;}
#print OUTP "Word ". $word. " " . $tf_val ." df: " . $df{$word}. "\n";
$weight[$i]{$word} = $tf_val * log($n / $df{$word}) / log(2);
#print "Weight ". $weight[$i]{$word}. " " . "\n";}}
# Compute weight of input phrase
foreach $word (sort keys %granddict) {
if (exists $inph{$word} ){
$tf_val = $inph{$word};}
else {$tf_val = 0;}
$inph_weight{$word} = $tf_val * log($n / $df{$word}) / log(2);
}# Step 4 Normalize the column of weights
```

```
#!perl -w
# matchline.pl
# Usage: matchline.pl <datafile><in phrase file ><outputfile>
# datafile must be present and a cmd line arg
# create a dictionary of all words in a file and alphabetize them
open(MYDATA, $ARGV[0]) or die("Error: cannot open file '$ARGV[0]'\n");
open(INPH, $ARGV[1]) or die("Error: cannot open file '$ARGV[1]'\n");
open(OUTP, ">$ARGV[2]") or die("Cannot open file '$ARGV[2]' for writing\n");
print OUTP "#Output results for ".$ARGV[0]."\n";
$nphr = 0;
# read in the file, get rid of newline and punctuation chars
while( $line = <MYDATA> ){
chomp($line);
$line =~ s/[-.?!"()'{}]//g;
@words = split(/ /,$line);
foreach $word (@words) {
++$granddict{lc($word)}; # this is the hash assignment lc is lowercase
++$tf[$nphr]{lc($word)};}
$nphr++;}
# Read in the input phrase from file
$linecnt = 0;
while( $line = <INPH> ){
print OUTP "Input Phrase: " . $line . "\n";
chomp($line);
$line =~ s/[-.?!"()'{}]//g;
@words = split(/ /,$line);
$linecnt++;}
# FIXME if ($linecnt != 1 ) die("Input phrase file must contain only 1 line");
print "inph linecount ". $linecnt . "\n";
foreach $word (@words){
++$inph{lc($word)};}
#print %tf[0];
# compute document frequencies
foreach $word (sort (keys(%granddict))){
$sum = 0;
for $i (0 .. $nphr) {
#print $word . "\n";
if ( exists $tf[$i]{$word} ) {
++$sum;}
}$df{$word} = $sum;
}# Step 3 Compute tf-idf weights
$n = $nphr + 1;
foreach $word (sort keys %granddict) {
for $i ( 0 .. $nphr) {
```

```
for $i ( 0 .. $nphr - 1){
$len2 = 0;
foreach $word ( sort keys %granddict){
$len2 += $weight[$i]{$word}**2;
#print $word ." len2 " . $len2 . "\n";
}$len = sqrt($len2);
foreach $word (sort keys %granddict ){
$unit[$i]{$word} = $weight[$i]{$word}/$len;
}}# Normalize input weight so it can be compared with the others
foreach $word (sort keys %granddict){
$len2 += $inph_weight{$word};
#print "inph ". $word ." len2 " . $len2 . "\n";

}$len = sqrt($len2);
foreach $word (sort keys %granddict ){
$inph_unit{$word} = $inph_weight{$word}/$len;
}#Step 5 Compute cosine simularities between input phrase and other phrases
$best = 0;
$best_idx = 0;
for $i ( 0 .. $nphr-1 ){
$sum = 0;
foreach $word (sort keys %granddict) {
$sum += $unit[$i]{$word} * $inph_unit{$word};
}$inph_cosine[$i] = $sum;
printf "INPH %d %.5f",$i, $inph_cosine[$i];
printf OUTP "INPH %d %.5f \n", $i, $inph_cosine[$i];
if ($inph_cosine[$i] > $best){
$best = $inph_cosine[$i];
$best_idx = $i;}}
printf "\nBest Match %.5f, %d\n", $best, $best_idx;
printf OUTP "\nBest Match %.5f, %d\n", $best, $best_idx;
# reopen the data file to get the best line since we didn't store it to save memory
open(MYDATA, $ARGV[0]) or die("Error: cannot open file '$ARGV[0]'\n");
$linecnt = 0;
while( $line = <MYDATA> ){
print $line . " linecount: ". $linecnt . "\n";
if( $linecnt == $best_idx ){
#print $line;
print OUTP "Best Line: " . $line . "\n";
$linecnt++;
last;}
else {$linecnt++;}
}# close the files
close MYDATA;
close INPH;
```

```
Libname disc
"C:\Users\MHE\Desktop\ActiveCourses\MKT.Eng";
/*the place the files would be put*/
run; /*always create library before importing*/
Proccontentsdata= disc.Discrim position;
Run;  /*check the data that is imported*/
/*analysis of description of the data*/
procmeansdata=disc.Discrim nmeanstdminmax;
var outdoor social conservative;
run;
procmeansdata=disc.Discrim nmeanstd;
class job;
var outdoor social conservative;
run;
proccorrdata=disc.Discrim;
var outdoor social conservative;
run;
procfreqdata=disc.Discrim;
tables job;
run;
/*discreminant analysis for specific dat"candisc" or
proc discrim*/
proccandiscdata=disc.Discrim out=discrim_out ;
class job;
var outdoor social conservative;
run;
/*figure of discriminant*/
data fakedata;
do outdoor = 0to30by1;
do social = 5to40by1;
do conservative = 0to25by1;
                  output;
end;
end;
end;
run;
procdiscrimdata=disc.Discrim testdata=fakedata
testout=fake_out out=discrim_out canonical;
class job;
var outdoor social conservative;
run;
data plotclass;
```

```
merge fake_out discrim_out;
run;
proctemplate;
define statgraph classify;
begingraph;
layout overlay;
contourplotparm x=Can1 y=Can2 z=_into_ / contourtype=fill

                                     nhint = 30 gridded = false;
scatterplot x=Can1 y=Can2 / group=job includemissinggroup=false
                                     markercharactergroup = job;
endlayout;
endgraph;
end;
run;
procsgrenderdata = plotclass template = classify;
run;
```

Text mining code in R (Re text mining &Data mining hand book)

Pang 2008[My General impression: really fruitful but untouched since people in marketing do not have skill, are reluctant to work under burden, are less empirical worker, so it is complete niche to be expert in and publish interesting papers]: predication it will become hot after skill availability for social network within 4 years. (Complexity of programming: recall compiler, not simple as SAS: mental efoort and intelligence), low spillover of computer experts(really nerd ones) to marketing

```
# preprocessing of the document
library(tm)
firefox.csv<-read.csv("c:/CDBookSurvay/Comments.csv")
firefox <- Corpus(DataframeSource(firefox.csv)) # create corpus for analysis
firefoxcopy <- firefox # keep a copy of corpus to use later as a dictionary for stem completion
firefox <-tm_map(firefox, tolower) # convert to lower case
firefox <- tm_map(firefox, removeNumbers) # remove numbers
for (j in 1:length(firefox)) firefox[[j]] <- gsub("'", " ",firefox[[j]])# to remove special puncutation but not connect
firefox <- tm_map(firefox, removePunctuation)# remove punctuations
firefox <- tm_map(firefox, removeWords, stopwords("english")) #remove stop words
newstopwords <- c("and", "for", "the", "to", "in", "when", "then", "he", "she", "than", "a", "for", "it", "of", "on", "to","im")
firefox <- tm_map(firefox, removeWords, newstopwords)

firefox <- tm_map(firefox, stemDocument)# stem words
inspect(firefox[1:10])
firefox <- tm_map(firefox, stemCompletion, dictionary=firefoxcopy) #stem completion

inspect(firefoxcopy[1:10])
summary(firefox)
myTdm <- TermDocumentMatrix(firefox, control = list(wordLengths=c(1,Inf)))
myTdm # printing dtm summery
idx <- which(dimnames(myTdm)$Terms =="alexa")

inspect(myTdm[idx+(0:5),1:10]) # look at 5 keywords after the keyword alexa over 10 documents that used for dtm
inspect(myTdm[0:20,1:10]) # check items of dtm
rownames(myTdm) # write all the keywords you have used
findFreqTerms(myTdm, lowfreq=3) #find frequent terms

# plot of more frequent words
termFrequency <- rowSums(as.matrix(myTdm)) # go over matrix and filtering for drawing a plot
termFrequency <- subset(termFrequency, termFrequency>=3) # go for terms that are in text more than 3 times
library(ggplot2) # use graphic package to draw plots
qplot(names(termFrequency), termFrequency, geom="bar") + coord_flip() # draw horizontal bar plot
barplot(termFrequency, las=2) # draw vertical bar plot
findAssocs(myTdm, "love", 0.25)# find words with highest asociation

library(wordcloud) # used for importance of the word check
m <- as.matrix(myTdm) # convert document term matrix to normal matrix
# calculate the frequency of words and sort it descendingly by frequency
wordFreq <- sort(rowSums(m), decreasing=TRUE)
# word cloud
set.seed(375) # to make it reproducible
grayLevels <- gray( (wordFreq+10) / (max(wordFreq)+10) )
# frequency below 1 is not ploted in the following
# random.order=F: frequent words plotted first in the center of the cloud
# set colour to: grayLevels or raingbow() to colorful or gray map
wordcloud(words=names(wordFreq), freq=wordFreq, min.freq=2, random.order=F,colors=grayLevels)

# clustering
# remove sparse terms
# you can remove sparce terms to avoid being flooded with words
myTdm2 <- removeSparseTerms(myTdm, sparse=0.95)
m2 <- as.matrix(myTdm2)
```

Opinion oriented information seeking
opinion mining and sentiment analysis
treatment of opinion, sentiment, and subjectivity in text,
summarization of evaluative text
how product or services perceived
classification of comments
response of firm after monitoring by modifying their marketingmessages, brand positioning, product development
query classification

Synonyms: opinion, view, belief, conviction, persuasion, sentiment mean a judgment one holds as true.

- opinion implies a conclusion thought out yet open to dispute (each expert seemed to have a different opinion.)
- view suggests a subjective opinion (very assertive in stating his views)
- belief implies often deliberate acceptance and intellectual assent (a firm belief in her party's platform.)
- conviction applies to a firmly and seriously held belief (the conviction that animal life is as sacred as human)
- persuasion suggests a belief grounded on assurance (as by evidence) of its truth (was of the persuasion thateverything changes)
- sentiment suggests a settled opinion reflective of one's feelings (her feminist sentiments are well-knowing)

Dave et al. 2003: "process a set of search results for a given item, generating a list of productattributes (***quality, features,*** etc.) and aggregating ***opinions*** about each of them (poor, mixed, good)"
Classifying reviews as to their polarity (either positive or negative).

Importance of opinion of others while decision making
Internet role that allows not acquitants and non professional critic, not colligue and not friend, people we never heard of opinion
Bias of rating of users and need correction

!! Question answering is another useful area
!! Context of the text (other advertising and things in the page that make people recall something
Detection of "flames" (overly-heated or antagonistic language)
Summarization for accounting for multiple view point
View bettered when includes more information
Why someone else is cited in the review (for literally reputation or supporting evidence?
computational treatment of affect

Subjective judgment of intangible qualities explanation of lack of purchase — e.g., "the design is tacky" or "customer service was condescending" — or even misperceptions —e.g., "updated device drivers aren't available"

1. creates condensed versions of individual reviews or adigest of overall consensus points
2. Idea about new product development: Market research from sources such as: Web — newsgroups, individual blogs, and aggregation sites such as Epinions
3. Besides reputation management and public relations. by tracking public viewpoints, one could perform trend prediction in sales or other relevant data
4. focus on what consumer are thinking
5. the issue of how ideas and innovations diffuse involves the question of who is positively or negatively disposed towards whom, and hence who would be more or less receptive to new information transmission from a given source
6. polarity of "ties" between people [54] and how this relates to group cohesion

Fundamental technology: Classification and extraction encompasses regression and ranking
Examples of problems this method used for:
1. **making a decision** for a particular phrase or document ("how positive is it?"), 2. **ordering a set of texts** ("rank these reviews by how positive they are"),
3**. giving a single label** to an entire document collection ("where on the scale between liberal and conservative do the writings of this author lie?"), and categorizing
4. The **relationship between two entities** based on textual evidence ("does A

```
# cluster of terms/words (come together e.g. couple of twits on text mining analysis, and couple of twits on job vacancies in PhD in different clusters)
distMatrix <- dist(scale(m2)) # calculate distance between terms after scaling
fit <- hclust(distMatrix, method="ward") # clustering agglomeration method is set to ward: icreased variance when two clusters are merged; other options are:  single linkage, complete linkage, average linkage, median and centroid
plot(fit)
# cut tree into 10 clusters
rect.hclust(fit, k=10) # cut into 10 clusters
(groups <- cutree(fit, k=10))

# clustering using k-min of documents
# transpose the matrix to cluster documents (tweets)
m3 <- t(m2) # take value of matrix as numeric & transpose to document term
# set a fixed random seed
set.seed(122) # to produce the clustering result
# k-means clustering of tweets
k <- 3 # 8 clusters
kmeansResult <- kmeans(m3, k)
# cluster centers
round(kmeansResult$centers, digits=3) # popular words in cluster and center

# check k mean cluster by top 3 words
for (i in 1:k) {
cat(paste("cluster ", i, ": ", sep=""))
s <- sort(kmeansResult$centers[i,], decreasing=T)
cat(names(s)[1:3], "\n")
# print the tweets of every cluster
# print(rdmTweets[which(kmeansResult$cluster==i)])
}

library(Rgraphviz)# to use for cluster assowciation matrix
plot(myTdm, terms = findFreqTerms(myTdm, lowfreq = 1)[1:20], corThreshold = 0)

library(fpc)#draw cluster based on matrix
plotcluster(m3, kmeansResult$cluster)

library(fpc) # clustering with Partitioning Around Medoids (PAM): (representative objects) more robust to noise and outliers than k-means clustering
# partitioning around medoids with estimation of number of clusters
pamResult <- pamk(m3, metric = "manhattan") # estimate number of optimal clusters
# number of clusters identified
(k <- pamResult$nc)
pamResult <- pamResult$pamobject
# print cluster medoids
for (i in 1:k) {
cat(paste("cluster", i, ": "))
cat(colnames(pamResult$medoids)[which(pamResult$medoids[i,]==1)], "\n")
#print tweets in cluster i
# print(rdmTweets[pamResult$clustering==i])
}

# plot clustering result
layout(matrix(c(1,2),2,1)) # set to two graphs per page
plot(pamResult, color=F, labels=4, lines=0, cex=.8, col.clus=1, col.p=pamResult$clustering)
layout(matrix(1)) # change back to one graph per page

#create social network of terms
termDocMatrix<-m2
termDocMatrix[1:5,1:5] # check Tdm
# change it to a Boolean matrix
termDocMatrix[termDocMatrix>=1] <- 1
# transform into a term-term adjacency matrix
termMatrix <- termDocMatrix %*% t(termDocMatrix) # %*% product of two matrices
# inspect terms numbered 5 to 7
termMatrix[5:7,5:7]
library(igraph)
# build a graph from the above matrix
g <- graph.adjacency(termMatrix, weighted=T, mode = "undirected")
# remove loops
g <- simplify(g)
# set labels and degrees of vertices
V(g)$label <- V(g)$name
V(g)$degree <- degree(g)
# set seed to make the layout reproducible
set.seed(3952)
layout1 <- layout.fruchterman.reingold(g)
plot(g, layout=layout1)

#dynamically rearranged layout get detail by running ?igraph::layout
plot(g, layout=layout.kamada.kawai)
tkplot(g, layout=layout.kamada.kawai)#extremely interesting graph creation
```

approve of B's actions?").

1. extraction problems (e.g., retrieving **opinions** on **various features** of a laptop) are often solved by **casting many sub-problems** as **classification** problems (e.g., given a text span, determine whether it expresses any opinion at all).
2. extraction is often a means to the further goal of **providing effective summaries** of the extracted information to users (combine information mined from multiple subjective text segments into a suitable summary)

Problem formulation and key concepts:
1. Sentiment polarity and **degrees of positivity** (locate its position on the continuum between these two polarities): sentiment-related classification/regression/ranking
binary categorization, multi-class categorization, regression, and/or ranking
2. Related categories: extract info on **why** reviewer liked or disliked the product "pros & cons"
3. Rating inference (ordinal regression): **multi-class** text categorization problem
Predicting degree of positivity provides more fine-grained rating information; **ordinal regression; mediocre**& neutral that is not strong feeling of good or bad (different from "lack of opinion"): reduce retaliation of seller, yet is **perceived neg**
4. Agreement detection: two text shall receive same or differing sentiment-related labels based on relationship b/w pairs?

- identification of subjectivity versus objectivity (effects of adjective orientation and gradability on sentence subjectivity: wiebe et. al): roots in studies in genre classification
- joint topic sentiment analysis: whether the document topic is related to subject of interest
- view point and perspective: more about attitude, n-ary classification
- various affect types six "universal" emotions: anger, disgust, fear, happiness, sadness, and surprise
- style analysis of the text
- feature vector or other representation that makes its most salient and important features 1.binary versus frequency based 2. Position at the beginning or end of document? (trigam and hierarchy) 3. Part of the speech: e.g. adjective 4. Syntax: e.g. modeling valence shifters such as negation, intensifiers, and diminishers 5. Negation "not, don't"; "I don't like deadlines", the token "like" is converted into the new token "like-NOT". Controversy when No does not negate: "No wonder this is considered one of the best".; problem of negation is more salient in sarcasm. E.g. "avoid" 6. Topic oriented features: PARTY will win", "go PARTY again", and "OTHER will win"

Approaches(Machine learning methods, mostly data mining methods (genetic, clustering, regression, …, but predictive usage), all are statistic methods; Not pure mechanical human should input into process):
- mapping a given piece of text, such as a document, paragraph, or sentence, to a label drawn from a pre-specified finite set or to a real number
- The impact of labeled data: Maximize entropy method
- Domain consideration: "unpredictable" is a positive description for a movie plot but a negative description for a car's steering abilities; 1. Use domain specific classifier
- Topic and subtopic: 1. on-topic text in the description or off topic , 2. Multiple topics

Unsupervised approaches:
- Classification using clustering technique
- Frequency of occurrence, prior polarity, cooccurance in the certain context
- Bootstraping: use the output of an available initial classifier to create labeled data, to which a supervised learning algorithm may be applied
- Classification based on relationship between documents: e.g. relationship between subdcouments or sentences. Degree of continuity (story telling), graph based techniques. "Respond to": when people respond to each other and that relationship which mostly has been antagonistic [addressing other person]
- Relationship between classes in contrast to multi-class categorization: 5-star is much similar to 4-star than 2-star.
- Discourse structure (overriding previous 4 line by simple sentence): e.g. [they] act wacky as hell...the ninth floor of hell...a cheap [beep] movie...The plotis such a mess that it's terrible. But I loved it.**[incorporating location of information is very important]**
- Identifying opinion holder, a person who does comparison and reveals her preference

Language models: topic relevancy, sentiment relevancy:
- difference in perspective upon the Kullback-Leibler(KL) divergence between posterior distributions induced from document collection pairs, and discover that the KL divergence between different aspects is an order of magnitude smaller than that between different topics.
- Probabilistic latent semantic analysis (PLSA) or latent Dirichlet allocation(LDA) can also be cast as language-modeling work The basic idea is to infer language models that correspond to **unobserved "factors**" in the data, with the hope that the factors that are learned represent topics or sentiment categories.

Oder of information algorithm:
Another way of capturing discourse structure information in documents is to model the global sentiment of a document as a trajectory of local sentiments. **Using sentiment flow as a sequential model** to represent an opinionated document. More specifically, each sentence in the document receives a local sentiment score from an isotonic-conditional-random-field-based sentence level predictor. The sentiment flow is defined as a function h : [0, 1) → O(the ordinal set), where the interval [(t − 1)/n, t/n) is mapped to the label of the t-th sentence in a document with n sentences.

```
pdf("term-network.pdf") # put terms plot in a pdf file
plot(g, layout=layout.fruchterman.reingold)
dev.off()

# size of plot's term according to the degree: important terms stand out
# set the width and transparency of edges based on their weights
# vertices and edges are accessed with V() and E()
# rgb(red, green, blue,alpha) defines a color with an alpha transparency
V(g)$label.cex <- 2.2 * V(g)$degree / max(V(g)$degree)+ .2
V(g)$label.color <- rgb(0, 0, .2, .8)
V(g)$frame.color <- NA
egam <- (log(E(g)$weight)+.4) / max(log(E(g)$weight)+.4)
E(g)$color <- rgb(.5, .5, 0, egam)
E(g)$width <- egam
# plot the graph in layout1
plot(g, layout=layout1)

#build network of documents (tweets) first phase
# remove "r", "data" and "mining" most used if they make the document crowded
# idx <- which(dimnames(termDocMatrix)$Terms %in% c("r", "data", "mining"))
#M <- termDocMatrix[-idx,] # remove terms from matrix
M<-termDocMatrix # since I did not wanted to remove anything
# build a tweet-tweet adjacency matrix
tweetMatrix <- t(M) %*% M
library(igraph)
g <- graph.adjacency(tweetMatrix, weighted=T, mode = "undirected")
V(g)$degree <- degree(g)
g <- simplify(g)
# set labels of vertices to tweet IDs
V(g)$label <- V(g)$name
V(g)$label.cex <- 1
V(g)$label.color <- rgb(.4, 0, 0, .7)
V(g)$size <- 2
V(g)$frame.color <- NA
barplot(table(V(g)$degree)) # check degree distribution of vertices

#build network of documents (tweets) second phase
idx <- V(g)$degree == 0
V(g)$label.color[idx] <- rgb(0, 0, .3, .7) # set based on degree
# set labels to the IDs and the first 10 characters of tweets
# limit to the first 20 character of every tweet
# label of each set to tweet ID so that graph would not be overcrowded
# set color and width of edges based on their weights
#V(g)$label[idx] <- paste(V(g)$name[idx], substr(df$text[idx], 1, 20), sep=" ")
egam <- (log(E(g)$weight)+.2) / max(log(E(g)$weight)+.2)
E(g)$color <- rgb(.5, .5, 0, egam)
E(g)$width <- egam
set.seed(3152)
layout2 <- layout.fruchterman.reingold(g)
plot(g, layout=layout2)

# remove isolated vertices and draw again
g2 <- delete.vertices(g, V(g)[degree(g)==0])
plot(g, layout=layout2)

# remove edges with low degree and draw again
g3 <- delete.edges(g, E(g)[E(g)$weight <= 1])
g3 <- delete.vertices(g3, V(g3)[degree(g3) == 0])
plot(g3, layout=layout.fruchterman.reingold)

# look at specific clique: considerably connected {replacement for dftext
inspect(firefox[c(15,16)])

#graph g directly from termDocMatrix
# create a graph
g <- graph.incidence(termDocMatrix, mode=c("all"))
# get index for term vertices and tweet vertices
nTerms <- nrow(M)
nDocs <- ncol(M)
idx.terms <- 1:nTerms
idx.docs <- (nTerms+1):(nTerms+nDocs)
# set colors and sizes for vertices
V(g)$degree <- degree(g)
V(g)$color[idx.terms] <- rgb(0, 1, 0, .5)
V(g)$size[idx.terms] <- 6
```

The flow is then smoothed out through convolution with a smoothing kernel. Finally, the distances between two flows (e.g., Lp distance between the two smoothed, continuous functions) should reflect, to some degree, the distances between global sentiments.

Sentiment without action is the ruin of the soul. — Edward Abbey
Romance should never begin with sentiment. It should begin with science and end with settlement. — Oscar Wilde, An Ideal Husband

Challenges:

1. determining which documents/portion are topically relevant to an opinion-oriented query
2. Quotation saying that it is from someone else
3. Summarizing the sentiment: Visualizing:

(a) **aggregation** of “votes” that may be registered on different scales (e.g., one reviewer usesa star system, but another uses letter grades)
(b) **selective highlighting** of some opinions
(c) representation of **points of disagreement** and points of consensus
(d) identification of **communities of opinion holders**
(e) accounting for different **levels of authority** among opinion holders

4. sentiment polarity text-classification: positive or negative:
the **inference and indirect sarcasm** sentence may not have negative word but imply negative:
my explanation: 1. not apply here since else people will have hard time understanding 2. It is limited domain with limited words 3. 20-80% as far as predicts sales and normal person understands it, it is good. 4. modern international people do not speak complicated (the targeted customer of this product), showing off their literature

5. **hypocritical** people say something like (I don’t want to talk about this), but they actually do

6. categorization of fact vs. opinion

7. previously loved but now hate (IMO: multiplication of positive and negative sense could work, prior and posterior; title and stars could be helpful in this sense)

8. abbreviations

9. product reviewer homophily with me in term of language conditional on I care (some people don’t care)

10. Sentiment and subjectivity are quite **context-sensitive**, and, at a **coarser granularity**, quite **domain dependent.** even the exact same expression can indicate different sentiment in different domains. (Go read the book in movie means negative sentiment but in book means good): IMO: **complementary product and substitute product mentioning**

11. the **order** in which different **opinions are presented** can result in a completely opposite overall sentiment polarity (in contrast to discourse analysis)

12. Course changing words such as “However”, “But”: my idea it does not change course completely but adds a second vector of negativity (like **hygiene parameter of working and incentive**): **and consider the asymmetric answer of humans to negative and positive information**

13. Order dependence of **comparisons: Comparison words finding & order analysis. Two category of words [+ vs. -] and then greater than or equal to: substitute products, complementary products.**

14. certainty vs. uncertainty (words : maybe, vs. must, will)

15. Past, present, future tense of the word (may not be really precise)

16. context that may make the difference for example **stock price rise is** a good news or bad?

17. objective information such as “long battery life”2 is often used to help determine the overall sentiment& whether this objective information is good or bad

18. The effect of specific words such as only: “the battery lasts 2 hours” vs “the battery **only** lasts 2 hours”& proximity of the meaning in the context: e.g. (“This laptop **only costs** $399”: how people judge attributes

19. determining degree of positivity: “The new model is more expensive than the old one” or “I prefer the new model to the old model”

20. identification of subjectivity versus objectivity (effects of adjective orientation and gradabilityon sentence subjectivity: wiebe et. al): roots in studiesin genre classification

21.various affect types six “universal” emotions: anger, disgust, fear, happiness, sadness, and surprise

22. Style analysis of the text and characteristics of a person

23. feature vector or other representation that makes its most salient and important features1.binary versus frequency based 2. Position at the beginning or end of document? (trigam and hierarchy) 3. Part of the speech: e.g. adjective 4. Syntax: e.g. modeling valence shifters such as negation, intensifiers, and diminishers 5. Negation “not, don’t”; “I don’t like deadlines”, the token “like” is converted into the new token “like-NOT”. Controversy when No does not negate: “No wonder this is considered one of the best”.; problem of negation is more salient in sarcasm. E.g. “avoid” 6. Topic oriented features: PARTY will win”, “go PARTY again”, and “OTHER will win”

```
V(g)$color[idx.docs] <- rgb(1, 0, 0, .4)
V(g)$size[idx.docs] <- 4
V(g)$frame.color <- NA
# set vertex labels and their colors and sizes
V(g)$label <- V(g)$name
V(g)$label.color <- rgb(0, 0, 0, 0.5)
V(g)$label.cex <- 1.4*V(g)$degree/max(V(g)$degree) + 1
# set edge width and color
E(g)$width <- .3
E(g)$color <- rgb(.5, .5, 0, .3)
set.seed(958)#5365, 227
plot(g, layout=layout.fruchterman.reingold)

# returns all vertices of "love" # if node does not exist returns "invalid vertex name"
V(g)[nei("love")]
V(g)[neighborhood(g, order=1, "love")[[1]]]# alternative way of geting vertices

#check which vertices include all three elements "thank", "perfect", "love"
(rdmVertices <- V(g)[nei("love") & nei("perfect") & nei("thank")])
inspect(firefox[as.numeric(rdmVertices$label)])# check content of the doc that includes these three terms

# remove three words to see the relationship with doc with other words
idx <- which(V(g)$name %in% c("love", "perfect", "thank"))
g2 <- delete.vertices(g, V(g)[idx-1])
g2 <- delete.vertices(g2, V(g2)[degree(g2)==0])
set.seed(209)
plot(g2, layout=layout.fruchterman.reingold)
```

My research

1 Time varying effect of comments on sales: comment window

2. product category and attributes

3. substitute and complimentary product mentioning

4. use Google keyword for relevant keywords and analysis

5. Use other available data sources as well to simplify this process of wording. There are many websites that you can capture content and take the intersection of sets or do the weighting.

5. product attribute such as (log in, theme, version, working, perceived risk)

6. Thesaurus through term-term matrix

7. Download source

8. Politeness "using f words, Damn)

9. Forward looking or myopic (hedonistic: emotion or utilitarian: attributes, profit,logic)

10. Whether the person asked the question, provides fact, or opinion

10. When model built run on other products as well and check the result [Cross category analysis: competing structure

Part that I skipped was summarization/ broader implication chapter, since I thought it did not really provide more information.