Homework 4

36-350: Data Mining

SOLUTIONS

1. *Similarity searching for cells*

   (a) *How many features are there in the raw data? Are they discrete or continuous? (If there are some of each, which is which?) If some are continuous, are they necessarily positive?*

   ANSWER: This depends on whether the cell classes are counted as features or not; you could defend either answer. If you don't count the cell class, there are 6830 continuous features. Since they are logarithms, they do not have to be positive. If you do count the class, there are 6831 features, one of which is discrete (the class).

   I am inclined not to count the clas as really being a feature, since when we get new data it may not have the label.

   (b) *Describe a method for finding the k cells whose expression profiles are most similar to that of a given cell. Carefully explain what you mean by "similar". How could you evaluate the success of the search algorithm?*

   ANSWER: The simplest approach would be to say that two cells are similar if their vectors of expression levels have a small Euclidean distance. The most similar cells are the ones whose epxression-profile vectors are closest, geometrically, to the target expression profile vector.

   There are several ways the search could be evaluated. One would be to use the given cell labels as an indication of whether the search results are "relevant". That is, one would look at the k closest matches and see (a) what fraction of them are of the same type as the target, which is the precision, and (b) what fraction of cells of that type are in the search results (the recall). Plotting the precision and recall against k gives an indication of how efficient the search is. This could then be averaged over all the labeled cells.

   Normalizing the vectors by the Euclidean length is possible here, though a little tricky because the features are the *logarithms* of the concentration levels, so some of them can be negative numbers, meaning that very little of that gene is being expressed. Also, the reason for normalizing bag-of-words vectors by length was the idea that two

documents might have similar meanings or subjects even if they had different sizes. A cell which is expressing all the genes at a low level, however, might well be very different from one which is expressing the same genes at a high level.

(c) *Would it make sense to weight genes by "inverse cell frequency"? Explain.*

ANSWER: No; or at least, I don't see how. Every gene is expressed to some degree by every cell (otherwise the *logarithm* of its concentration would be $-\infty$), so every gene would have an inverse cell frequency of zero.

However, a related idea would be to scale genes' expression levels by something which indicates how much they *vary* across cells, such as the range or the variance. Since we have labeled cell types here, we could even compute the average expression level for each type and then scale genes by the variance of these type-averages. That is, genes with a small variance (or range, etc.) should get small weights, since they are presumably uninformative, and genes with a large variance in epxression level should get larger weights.

2. *The file* `nci.kmeans` *records the true cell type and the result of running the k-means algorithm on the data three different times, each time with* $k = 14$. *(The R command for this is* `kmeans`*.)*

   (a) *For each run, calculate the number of errors made by k-means, i.e., how many pairs of cells of the same type are in different clusters, and how many pairs of cells of different types are in the same cluster.*

   ANSWER: This *can* be done by hand, but it's easiest to do with some code. For each *pair* of cells, we need to check whether they have the same true class but are in different clusters (call that a type I error), or have different true classes but are in the same cluster (call that a type II error). Note that we only have to check *distinct* pairs of cells — so having checked whether cell 1 and cell 2 give us an error, we don't have to check again for cell 2 and cell 1. Nor do we have to check cell 1 against cell 1.

   Here is the most straightforward way to do it. (A more R-ish solution would replace the two explicit `for` loops with vectorized calculations.)

```
count.clustering.errors <- function(true.classes,inferred.clusters) {
  # Should double-check that the two input vectors have the same length!
  n = length(true.classes)
  # Always a good idea to explicitly declare counters
  type.i.errors = 0
  type.ii.errors = 0
  for (i in 1:(n-1)) {
    for (j in (i+1):n) {
      if ((true.classes[i] == true.classes[j])
          & (inferred.clusters[i] != inferred.clusters[j])) {
        type.i.errors = type.i.errors +1
      }
      if ((true.classes[i] != true.classes[j])
          & (inferred.clusters[i] == inferred.clusters[j])) {
        type.ii.errors = type.ii.errors +1
      }
    }
  }
  total.errors = type.i.errors + type.ii.errors
  n.distinct.pairs = n*(n-1)/2
  total.error.rate = total.errors/n.distinct.pairs
  return(list(type.i.errors=type.i.errors,type.ii.errors=type.ii.errors,
              total.errors=total.errors,total.error.rate=total.error.rate)
}
```

   The problem statement just asks for a *count* of errors. The function also calculates the error rate, i.e., the ratio of mis-clustered pairs to the total number of pairs. (That total is $n(n-1)/2$ because there are n choices for the first cell in the pair, followed by $n-1$ choices

for the second pair; but this counts each pair twice, once as i, j and once as j, i, so we need to divide by two.)

Let's try this out before using this for a real answer. The first command loads them in a data frame. (The `skip=2` argument tells it to skip the first two lines, then `header=TRUE` lets it know that the first line to read names the columns, rather than containing actual data.) The second one checks the dimensions of the data frame to make sure everything worked (it did — 64 rows by 4 columns). Then I print out the first four rows. In this case I can work out the right answer by hand: cells 1 and 2 are correctly groups together by the `clusterA` results, but cells 1 and 2 are incorrectly separated from cell 3 — so that's two type I errors. Cell 3 is in turn incorrectly clustered together with cell 4, a type II error. So if I run my function on the first four columns of `clusterA` column, I should get a 2 type I errors and 1 type II error.

```
> nci.kmeans = read.table("nci.kmeans",skip=2,header=TRUE)
> dim(nci.kmean) # check for sane dimensions
[1] 64  4
> nci.kmeans[1:4,]
   type clusterA clusterB clusterC
1   CNS       11        3       12
2   CNS       11        3       12
3   CNS       13        3       12
4 RENAL       13        3        9
> count.clustering.errors(nci.kmeans$type[1:4],nci.kmeans$clusterA[1:4])
$type.i.errors
[1] 2
$type.ii.errors
[1] 1
$total.errors
[1] 3
$total.error.rate
[1] 0.5
```

Having gotten some confidence in the function, let's run it for real on each column.

```
> count.clustering.errors(nci.kmeans$type,nci.kmeans$clusterA)
$type.i.errors
[1] 100
$type.ii.errors
[1] 97
$total.errors
[1] 197
$total.error.rate
[1] 0.09771825
```

Summarizing the results for all three runs

Run	Type I	Type II	Total
A	100	97	197
B	106	79	185
C	109	182	191
Mean± sd	105 ± 5	119 ± 55	$191pm6$

(b) *Is one kind of error more frequent in all three runs?*

ANSWER: Type I errors — putting cells which are in the same class into different clusters — are more common than type II errors (other way around).

(c) *Are there any classes which seem particularly hard for k-means to pick up?*

ANSWER: It's possible to answer this question impressionistically, but also quantitatively.

Start by making a table which counts how often cells of each type appear in each cluster.

```
> table(nci.kmeans$type,nci.kmeans$clusterA)

          1 2 3 4 5 6 7 8 9 10 11 12 13 14
  BREAST  0 0 0 0 2 0 0 0 2  0  0  0  2  1
  CNS     0 0 0 0 0 0 0 0 0  0  2  0  3  0
  COLON   0 0 0 1 0 0 0 0 0  0  0  6  0  0
  K562A   0 0 0 0 0 0 0 0 1  0  0  0  0  0
  K562B   0 0 0 0 0 0 0 0 1  0  0  0  0  0
  LEUKEMIA 0 0 0 0 0 0 4 0 2  0  0  0  0  0
  MCF7A   0 0 0 0 0 0 0 0 1  0  0  0  0  0
  MCF7D   0 0 0 0 0 0 0 0 1  0  0  0  0  0
  MELANOMA 1 0 6 0 0 1 0 0 0  0  0  0  0  0
  NSCLC   0 0 0 4 0 1 0 1 0  2  0  0  0  1
  OVARIAN  0 0 0 5 0 1 0 0 0  0  0  0  0  0
  PROSTATE 0 0 0 2 0 0 0 0 0  0  0  0  0  0
  RENAL   0 7 0 0 0 1 0 0 0  0  0  0  1  0
  UNKNOWN  0 0 0 0 0 1 0 0 0  0  0  0  0  0
```

A cell type is easy for clustering to detect if it always, or for the most part, appears in one the same cluster. Looking at the table, for instance, it looks like renal cancer is fairly easy to detect as a cluster (7 of 9 cases in cluster 2), but breast cancer is harder (spread evenly over four clusters). Let's divide the counts in each row by the total number of cells in that row — this will give us more sense of how concentrated the results are in different clusters

```
> round(t(apply(table(nci.kmeans$type,nci.kmeans$clusterA),1,
        function(x){x/sum(x)})),digits=2)
```

```
             1    2    3    4    5    6    7    8    9   10  11   12   13   14
  BREAST   0.00 0.00 0.00 0.00 0.29 0.00 0.00 0.00 0.29 0.00 0.0 0.00 0.29 0.14
  CNS      0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.4 0.00 0.60 0.00
  COLON    0.00 0.00 0.00 0.14 0.00 0.00 0.00 0.00 0.00 0.00 0.0 0.86 0.00 0.00
  K562A    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.0 0.00 0.00 0.00
  K562B    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.0 0.00 0.00 0.00
  LEUKEMIA 0.00 0.00 0.00 0.00 0.00 0.00 0.67 0.00 0.33 0.00 0.0 0.00 0.00 0.00
  MCF7A    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.0 0.00 0.00 0.00
  MCF7D    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.0 0.00 0.00 0.00
  MELANOMA 0.12 0.00 0.75 0.00 0.00 0.12 0.00 0.00 0.00 0.00 0.0 0.00 0.00 0.00
  NSCLC    0.00 0.00 0.00 0.44 0.00 0.11 0.00 0.11 0.00 0.22 0.0 0.00 0.00 0.11
  OVARIAN  0.00 0.00 0.00 0.83 0.00 0.17 0.00 0.00 0.00 0.00 0.0 0.00 0.00 0.00
  PROSTATE 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.0 0.00 0.00 0.00
  RENAL    0.00 0.78 0.00 0.00 0.00 0.11 0.00 0.00 0.00 0.00 0.0 0.00 0.11 0.00
  UNKNOWN  0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.0 0.00 0.00 0.00
```

So the type BREAST is not well-clustered, nor is CNS (almost evenly split between two clusters), and NSCLC is especially bad. Types K562A, K562B, MCF7A, MCF7D and Unknown seem well-clustered, but there's only one example of each so that hardly counts.

Another way to do this is to look at the entropy of the inferred clusters for each cell type. First define an entropy function:

```
entropy.from.counts <- function(x) {
# Normalize to get a distribution
p = x/sum(x)
# Discard the zero-probability entries
p = p[p>0]
entropy=-sum(p*log(p,base=2))
return(entropy)
}
```

Now apply it to the cross-classification table.

```
> round(apply(table(nci.kmeans$type,nci.kmeans$clusterA),1,entropy.from.counts),2)
  BREAST      CNS    COLON    K562A    K562B LEUKEMIA    MCF7A    MCF7D MELANOMA
    1.95     0.97     0.59     0.00     0.00     0.92     0.00     0.00     1.06
 OVARIAN PROSTATE    RENAL  UNKNOWN
    0.65     0.00     0.99     0.00
```

This is just looking at clusterA results, but we can average the entropies across the three clustering runs.

```
> table.A = table(nci.kmeans$type,nci.kmeans$clusterA)
> entropies.A = apply(table.A,1,entropy.from.counts)
> table.B = table(nci.kmeans$type,nci.kmeans$clusterB)
> entropies.B = apply(table.B,1,entropy.from.counts)
> table.C = table(nci.kmeans$type,nci.kmeans$clusterC)
> entropies.C = apply(table.C,1,entropy.from.counts)
```

```
> clustering.entropies = rbind(entropies.A,entropies.B,entropies.C)
> apply(clustering.entropies,2,mean)
   BREAST       CNS     COLON     K562A     K562B  LEUKEMIA     MCF7A     MCF7D
1.9502121 0.8879431 0.5916728 0.0000000 0.0000000 1.1622047 0.0000000 0.0000000
MELANOMA    NSCLC   OVARIAN  PROSTATE     RENAL   UNKNOWN
1.0612781 1.9847398 0.8505580 0.0000000 1.2418562 0.0000000
```

So, the BREAST and NSCLC types are the hardest to cluster according to this, followed by RENAL and LEUKEMIA.

(d) *Are there any pairs of cells which are always clustered together, and if so, are they of the same class?*

ANSWER: There are many cells which are always clustered together (at least in these three runs), but they are not always of the same class. Cells 1 and 2 are clustered together and are of the same class; cells 4 and 5 are clustered together but are of different classes.

3. *Figures 1 and 2 show hierarchical clusterings of the data. (The R command used was* `hclust`*, with* `method` *set to* `ward` *or* `single`*.)*

   (a) *Which cell classes seem to be best captured by each clustering method?*

   ANSWER: Ward's method does well on MELANOMA (near middle of plot) and COLON (near bottom). There are also some sub-clusters of OVARIAN, BREAST and NSCLC. The single-link method again does well with MELANOMA and COLON.

   (b) *Figure 3 shows the merging-cost curve for Ward's method. How many clusters does this suggest?*

   ANSWER: There are peaks around $k = 40$, $k = 20$ and $k = 10$.

   (c) *Which method does a better job of capturing the cell classes? (Justify your answer.)*

   ANSWER: The single-link method *looks* better — scanning down the figure, a lot more of the adjacent cells are of the same type. Since the dendrogram puts items in the same sub-cluster together, this suggests that the clustering more nearly corresponds to the known cell types.

   (d) *Suppose you did not know the cell classes. Can you think of any reason to prefer one clustering method over another here, based on their outputs and the rest of what you know about the problem?*

   ANSWER: I can't think of anything very compelling.

4. *Figure 4 shows the fraction of the total variance (R^2) described by the first k principal components, as a function of k.*

   (a) *How (if at all) is this different from a scree plot?*
   ANSWER: This shows the cumulative sum of the eigenvalues, rather than the eigenvalues themselves. The scree plot would be obtained by taking the differences between successive pointsin this figure.

   (b) *Roughly how much variance is captured by the first two principal components?*
   ANSWER: A bit more than 20% of the variance, say about 25%.

   (c) *Roughly how many components must be used to capture half of the variance? To capture nine-tenths?*
   ANSWER: 10; 30.

   (d) *Is there any point to using more than 50 components? (Justify your answer.)*
   ANSWER: There's very little point from the point of view of capturing variance. The error in reconstructing the expression levels from using 50 components is already extremely small. However, there's no guarantee that, in some other application, the information carried by the 55th component (say) wouldn't be crucial.

   (e) *How much confidence should you have in results from visualizing the first two principal components?*
   ANSWER: Very little; the first two components represent only a small part of the total variance.

5. *Figure 5 shows the projection of the 64 cells on to the first two principal components.*

   (a) *One tumor class (at least) forms a cluster in the projection. Say which, and explain your answer.*
   ANSWER: MELANOMA; most of the cells cluster in the bottom-center of the figure (PC1 about 0, PC2 between -2 and -8), with only a few non-MELANOMA cells among them, and a big gap to the rest of the data. One could also argue for LEUKEMIA in the upper left.

   (b) *Identify a tumor class which does* not *form a compact cluster.*
   ANSWER: Most of them don't. What I had in mind though was BREAST, which is very widely spread.

   (c) *Of the two classes of tumors you have just named, which will be more easily classified with the prototype method? With the nearest neighbor method?*
   ANSWER: BREAST will be badly classified using the prototype method. The prototype will be around the middle of the plot, where there are no breast-cancer cells. Nearest-neighbors can hardly do worse. On the other hand, MELANOMA should work better with the prototype method, because it forms a compact blob.

6. *The file* `nci.pca2.kmeans` *contains the results of running the k-means algorithm three times on the PCA-projected data, with* $k = 14$. *(As the name indicates, this used just the first two principal components.)*

   (a) *Calculate the number of errors, as with the k-means clustering based on all 100 genes.*

   ANSWER:

```
> nci.pca2.kmeans = read.table("nci.pca2.kmeans",skip=2,header=TRUE)
> count.clustering.errors(nci.pca2.kmeans$type,nci.pca2.kmeans$clusterA)
$type.i.errors
[1] 115
$type.ii.errors
[1] 79
$total.errors
[1] 194
```

   Summarizing, here are the results over all three cases.

Run	Type I	Type II	Total
A	115	79	194
B	125	94	219
C	142	146	288
Mean $\pm$ sd	127 ± 13	106 ± 35	233 ± 47

   For comparison, the counts of type I and type II errors obtained using the full set of genes were 105 ± 5 and 119 ± 55. So by eliminating most of the features we have increased the type I error rate a little, with basically no change to the type II error rate.

   (b) *Are there any pairs of cells which are always clustered together? If so, do they have the same cell type?*

   ANSWER: Cells 1 and 3 are always clustered together, with type CNS. Cells 4 and 5, BREAST and RENAL, are also always clustered together.

   (c) *Does k-means find a cluster corresponding to the cell type you thought would be especially easy to identify in the previous problem? (Explain your answer.)*

   ANSWER: Cluster 9 in run A and cluster 5 in run B consist exclusively of MELANOMA cells. However, there are some MELANOMA cells outside this cluster, and in run C it gets split into two clusters (numbers 4 and 8). Still, this is a pretty good match, as might be excepted.

   (d) *Does k-means find a cluster corresponding to the cell type you thought be be especially hard to identify?*

   ANSWER: No. If we do a table again,

```
> table(nci.pca2.kmeans$type,nci.pca2.kmeans$clusterA)
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
BREAST	2	2	0	2	0	0	0	0	0	0	0	0	0	1
CNS	0	0	0	0	0	0	0	0	0	5	0	0	0	0
COLON	0	0	0	0	1	0	0	0	0	0	0	0	6	0
K562A	0	0	0	0	0	0	0	0	0	0	1	0	0	0
K562B	0	0	0	0	0	0	0	0	0	0	1	0	0	0
LEUKEMIA	0	0	0	0	0	2	1	0	0	0	3	0	0	0
MCF7A	0	0	0	1	0	0	0	0	0	0	0	0	0	0
MCF7D	0	0	0	1	0	0	0	0	0	0	0	0	0	0
MELANOMA	0	1	0	0	0	0	0	0	6	0	0	1	0	0
NSCLC	1	0	1	0	0	0	3	0	0	1	0	2	0	1
OVARIAN	0	0	4	0	0	0	0	1	0	0	0	1	0	0
PROSTATE	0	0	0	0	0	0	0	0	0	0	0	1	1	0
RENAL	1	0	0	0	0	0	0	2	0	0	0	5	0	1
UNKNOWN	0	0	0	0	0	0	0	1	0	0	0	0	0	0

we see that BREAST is spread (in this run) over 4 clusters, and doesn't even dominate any of those clusters. So it doesn't correspond to a cluster.

7. *Install the library* `ElemStatLearn` *from CRAN and call up the data set with the command* `data(nci)`*. This is set up so that each* column *is a different cell, and the column names are the cell classes. You will want to transpose this so that the cells are rows and the genes are columns.*

   (a) *Use* `prcomp` *to do a principal components analysis of the* complete *set of genes. What are the first eight eigenvalues? Should you set* `scale.=TRUE`*? Explain your reasoning.*

   Answer: Here's how I do the PCA.

```
> library(ElemStatLearn)
> data(nci)
> nci.t = t(nci)
> nci.pca = prcomp(nci.t)
> nci.pca.scaled = prcomp(nci.t,scale.=TRUE)
> nci.pca$sdev[1:8]^2
[1] 633.2156 352.9278 279.9189 183.0830 163.5573 149.0968 122.2882 119.7912
> nci.pca.scaled$sdev[1:8]^2
[1] 775.8157 461.4486 392.8508 290.1080 255.0986 247.1524 209.4230 183.4472
```

   As for whether or not to use scaling, both answers are defensible. To argue against scaling, notice that:

   - the units in which all the features are measured here are comparable (log molecular concentrations), unlike some of the data sets where we'd be comparing feet to dollars, or days of frost to homicides;
   - without scaling, while the first eigenvector is larger than the others (naturally!), it's not overwhelmingly larger
   - The variances of the individual features are mostly within a fairly small range, and even the hugest of them isn't vastly larger than the others.

```
> summary(apply(nci.t,2,var))
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
 0.04056  0.20890  0.31930  0.62250  0.64470 11.72000
```

   To argue for scaling, notice that

   - there are a few features with variances a few times larger or smaller than the rest, and they could cause trouble; why give them the opportunity?
   - the differences between cells may depend on whether the lie in each gene's expression spectrum, so even though the units are physically comparable across features, a 1 unit change in feature 7 may not have the same importance as a 1 unit change in feature 563.

   (b) *Write a function to print cell class labels against the projections on to the first two components. How different does the output look from Figure 5? Are the same clusters visible in your plot as in the figure?*

Answer:

Here is a solution based on the code in Lecture 14.

```
plot.labels.pca.1 <- function(fitted.pca,...) {
  x = fitted.pca$x[,1]
  y = fitted.pca$x[,2]
  labels = rownames(fitted.pca$x)
  plot.new()
  plot.window(xlim=range(x),ylim=range(y))
  text(x,y,labels,...)
  axis(1)
  axis(2)
}
```

The `...` lets me pass extra graphical arguments to the text-plotting function `text`. For instance, I can control the size of the labels with the `cex` argument (Figure 6).

```
> plot.labels.pca.1(nci.pca,cex=0.75)
```

Much the same effect can be had changing the options on the `biplot` function.

```
biplot(nci.pca,cex=c(1,0.001),expand=0.00001)
```

Here the `cex=c(1,0.001)` argument tells R to shrink the text for the labels of the feature-arrows by a factor of 1000. Similarly the length of the feature-arrows is to be expanded by a factor of 0.00001 (i.e., shrunk 100,000 times). The result is a very small red dot just visible in the center of the plot. It does, however, take a while, since R is still drawing and labeling 6830 minute arrows. I omit this figure from the solutions, because it takes forever to actually print, though R will render it on the screen in a minute or two.

(c) *Calculate the error rate of the prototype method using* all *the gene features, using the leave-one-out estimate. You may use the solution code from homework 2, or your own code. (Note: this may take several minutes to run. [Why so slow?])*

Answer:

```
> leave.one.out.error(nci.t,rownames(nci.t),method="prototypes",
                      preproc=FALSE)
$error.rate
[1] 0.421875
```

I set `preproc=FALSE` because I do not want to use inverse "document" frequency or Euclidean length normalization.

The calculation takes 10 minutes on my computer. The reason is that my code isn't optimized. When I hold back a data point, that changes the prototype for only *one* class. However, to make the code

simpler, my function re-calculates the prototypes for *all* classes every time. Since there are 14 classes, 6830 features, and 64 cells, it ends up doing $14 * 6830 * 64 = 6{,}119{,}680$ averages. A more efficient program would calculate prototypes for all classes with all data ($14 * 6830 = 95{,}620$ averages), and then adjust the prototype only for the class of the held-out cell ($64 * 6830 = 437{,}120$ subtractions).

(d) *Calculate the error rate of the prototype method using the first two, ten and twenty principal components. Include the R commands you used to do this.*

ANSWER: To call the `leave.one.out.error` function, we need a data matrix and a vector of classes.

```
> leave.one.out.error(nci.pca$x[,1:2],rownames(nci.t),method="prototypes",
                      preproc=FALSE)
$error.rate
[1] 0.578125
```

`nci.pca$x[,1:2]` gives the first two columns of PCA scores for each cell. I set `preproc=FALSE` because I don't want to use Euclidean length normalization or inverse "document" frequency weighting. (The function also provides the row-numbers of the data points where the classifier makes an error, but I've left them out here.) Similarly

```
> leave.one.out.error(nci.pca$x[,1:10],rownames(nci.t),method="prototypes",
                      preproc=FALSE)$error.rate
[1] 0.484375
> leave.one.out.error(nci.pca$x[,1:20],rownames(nci.t),method="prototypes",
                      preproc=FALSE)$error.rate
[1] 0.40625
```

Notice that with 20 principal components we have a better error rate than if we used the whole data.

If I use the scaled PCA instead, I get error rates of 0.67, 0.5 and 0.41 instead.

(e) *(Extra credit.) Plot the error rate, as in the previous part, against the number q of principal components used, for q from 2 to 64. Include your code and comment on the graph. (*Hint*: Write a function to calculate the error rate for arbitrary q, and use* `sapply`*.)*

ANSWER: Take what we did in the last part and turn it into a function.

```
proto.error.q = function(q) {
  first.q.pcs = nci.pca$x[,1:q]
  true.classes = rownames(nci.pca$x)
  L1OE = leave.one.out.error(first.q.pcs,true.classes,
                             method="prototypes",
                             preproc=FALSE)
```

```
  return(L1OE$error.rate)
}
```

Quick check of whether this works:

```
> proto.error.q(2)
[1] 0.578125
> proto.error.q(10)
[1] 0.484375
```

These match what we saw in the previous part, so the function seems to be working properly.

Now calculate it for all q in the range, and plot (Figure 7).

```
error.rate.vs.pcs = sapply(2:64,proto.error.q)
plot(2:64,error.rate.vs.pcs,
     xlab="number of principal components",ylab="error rate",type="b",
     ylim=c(0,0.6))
abline(h=0.421875,lty=2)
```

The dashed line shows the error rate when using all the features. Notice that this is larger than the error rate with ≈ 15 principal components.

If you scaled the features before extracting the principal components, the same solution works.

```
proto.error.scaled.q = function(q) {
  first.q.pcs = nci.pca.scaled$x[,1:q]
  true.classes = rownames(nci.pca.scaled$x)
  L1OE = leave.one.out.error(first.q.pcs,true.classes,
                             method="prototypes",
                             preproc=FALSE)
  return(L1OE$error.rate)
}
```

(Since the code is almost exactly the same, it would be better practice to change the function to take *two* arguments, q and the PCA scores matrix, rather have two functions.)

```
points(2:64,sapply(2:64,proto.error.scaled.q),pch=23)
```

This adds the new points to the existing plot, but marks the points with diamonds instead of circles. (Calculating the data-points in the plotting command itself means less code, and fewer objects in the workspace, but it also means extra time re-plotting, so I don't recommend it unless you're sure you've got all your graphics options nailed down.) Scaling leads to very similar error rates, though generally somewhat higher, especially in the region $q \approx 15$.

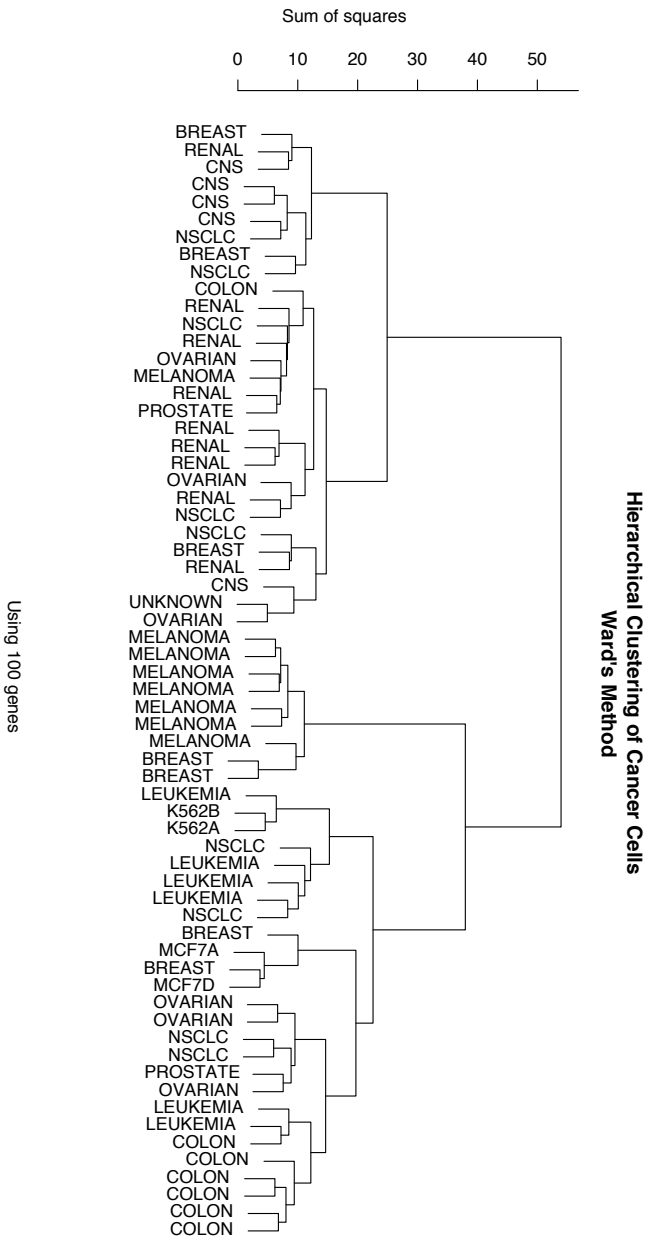
Sum of squares
0
10
20
30
40
50
BREAST
RENAL
CNS
CNS
CNS
CNS
NSCLC
BREAST
NSCLC
COLON
RENAL
NSCLC
RENAL
OVARIAN
MELANOMA
RENAL
PROSTATE
RENAL
RENAL
RENAL
OVARIAN
RENAL
NSCLC
NSCLC
BREAST
RENAL
CNS
UNKNOWN
OVARIAN
MELANOMA
MELANOMA
MELANOMA
MELANOMA
MELANOMA
MELANOMA
MELANOMA
BREAST
BREAST
LEUKEMIA
K562B
K562A
NSCLC
LEUKEMIA
LEUKEMIA
LEUKEMIA
NSCLC
BREAST
MCF7A
BREAST
MCF7D
OVARIAN
OVARIAN
NSCLC
NSCLC
PROSTATE
OVARIAN
LEUKEMIA
LEUKEMIA
COLON
COLON
COLON
COLON
COLON
COLON
Hierarchical Clustering of Cancer Cells
Ward's Method
Using 100 genes

Figure 1:

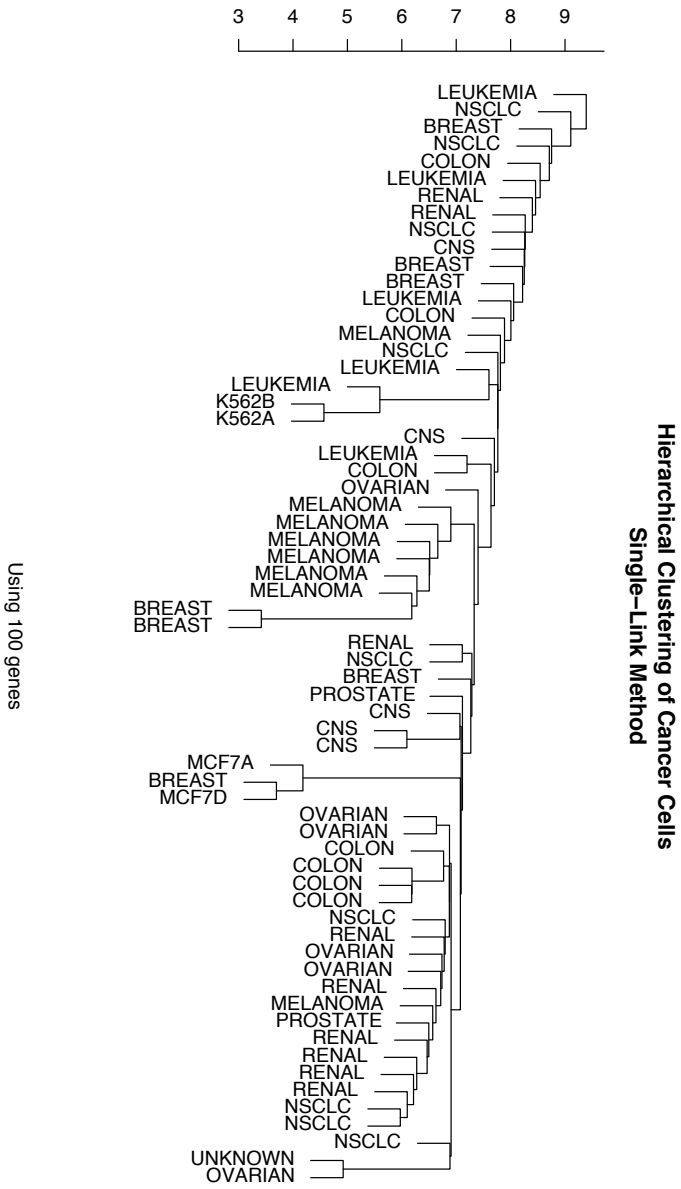
Hierarchical Clustering of Cancer Cells
Single−Link Method
Using 100 genes
3
4
5
6
7
8
9
LEUKEMIA
NSCLC
BREAST
NSCLC
COLON
LEUKEMIA
RENAL
RENAL
NSCLC
CNS
BREAST
BREAST
LEUKEMIA
COLON
MELANOMA
NSCLC
LEUKEMIA
LEUKEMIA
K562B
K562A
CNS
LEUKEMIA
COLON
OVARIAN
MELANOMA
MELANOMA
MELANOMA
MELANOMA
MELANOMA
MELANOMA
BREAST
BREAST
RENAL
NSCLC
BREAST
PROSTATE
CNS
CNS
CNS
MCF7A
BREAST
MCF7D
OVARIAN
OVARIAN
COLON
COLON
COLON
COLON
NSCLC
RENAL
OVARIAN
OVARIAN
RENAL
MELANOMA
PROSTATE
RENAL
RENAL
RENAL
RENAL
NSCLC
NSCLC
NSCLC
UNKNOWN
OVARIAN

Figure 2:

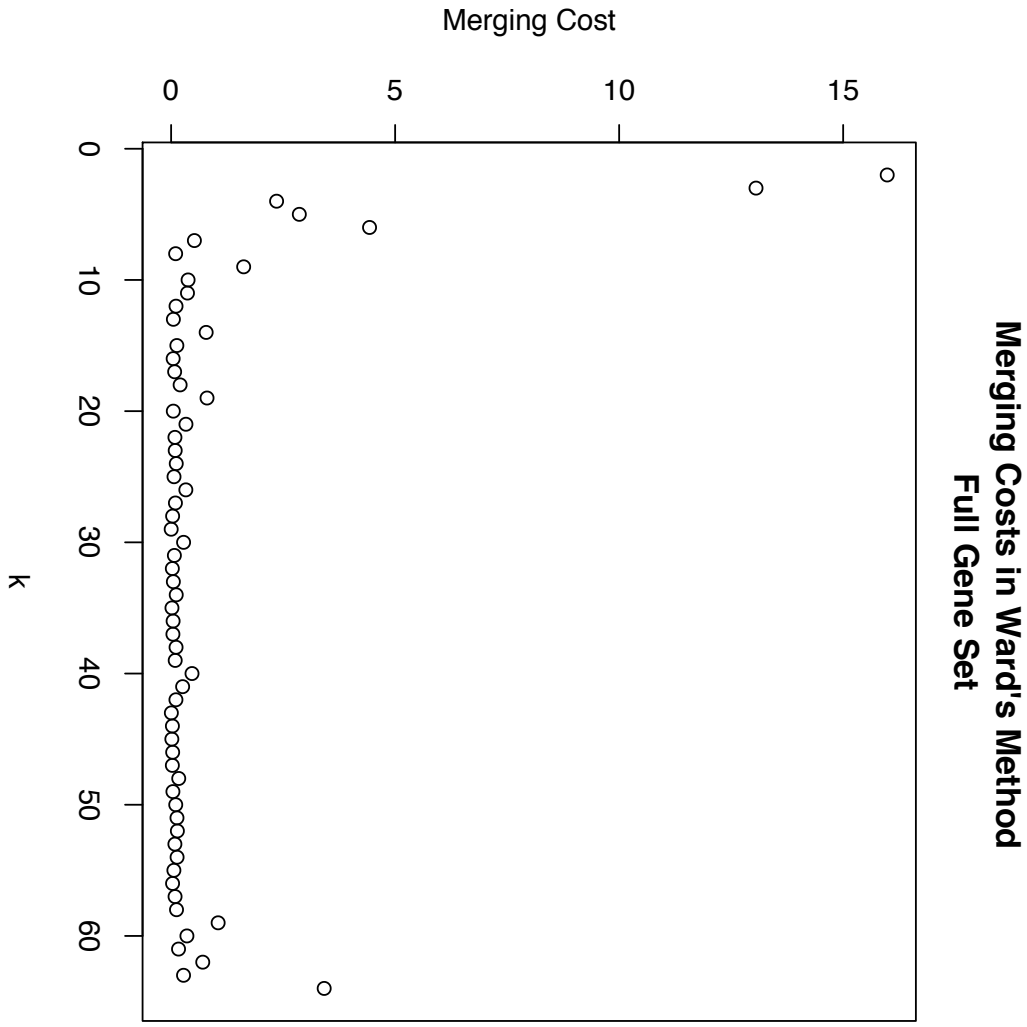
Merging Cost
0
5
10
15
k
0
10
20
30
40
50
60
Merging Costs in Ward's Method
Full Gene Set

Figure 3:

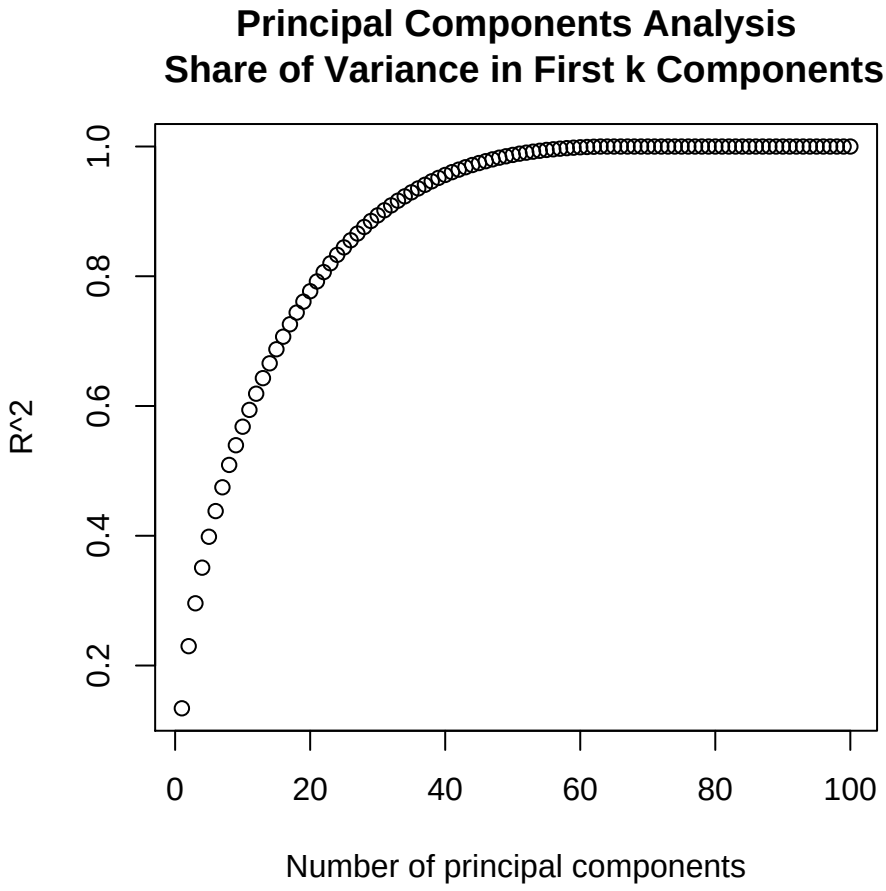
Principal Components Analysis
Share of Variance in First k Components
R^2
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100
Number of principal components

Figure 4:

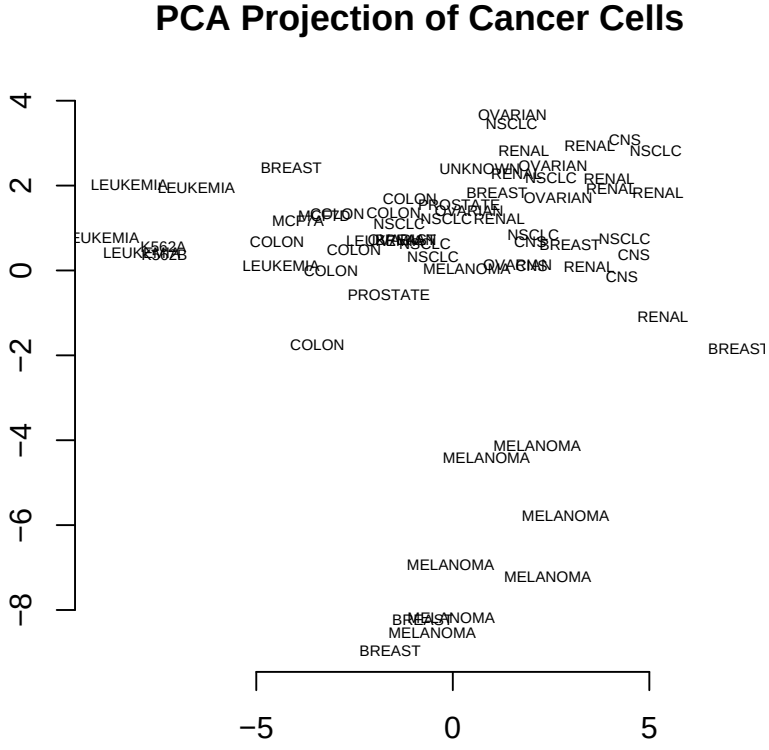
PCA Projection of Cancer Cells
4
2
0
−2
−4
−6
−8
−5
0
5
OVARIAN
NSCLC
RENAL
RENAL
CNS
NSCLC
BREAST
UNKNOWN
OVARIAN
RENAL
NSCLC
RENAL
LEUKEMIA
LEUKEMIA
BREAST
OVARIAN
RENAL
RENAL
COLON
PROSTATE
MCF7A
NSCLC
NSCLC
RENAL
NSCLC
COLON
BREAST
NSCLC
COLON
CNS
LEUKEMIA
COLON
MELANOMA
RENAL
CNS
PROSTATE
RENAL
COLON
BREAST
MELANOMA
MELANOMA
MELANOMA
MELANOMA
MELANOMA
BREAST
MELANOMA
BREAST

Figure 5:

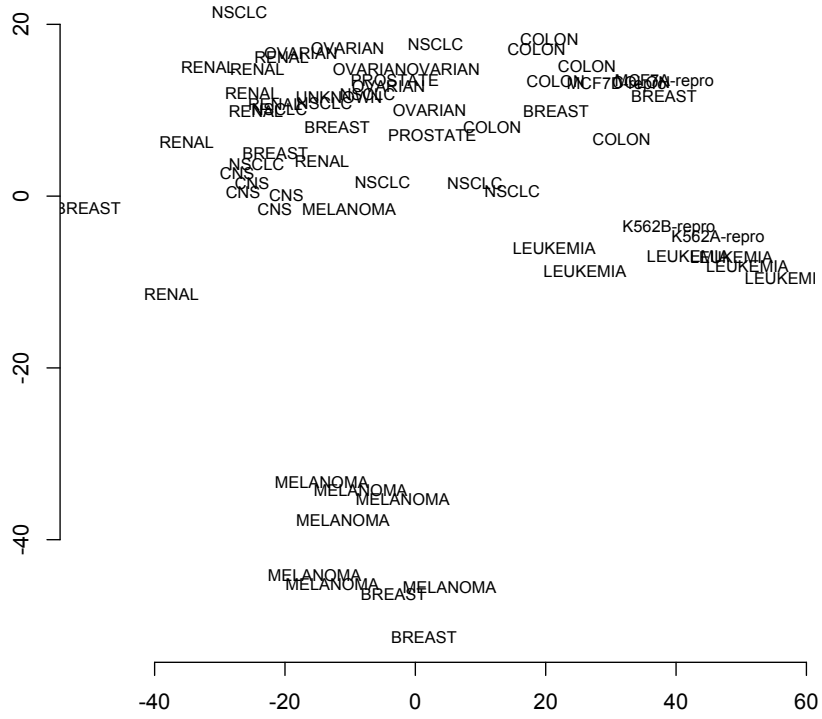

Figure 6: First solution to problem 7a

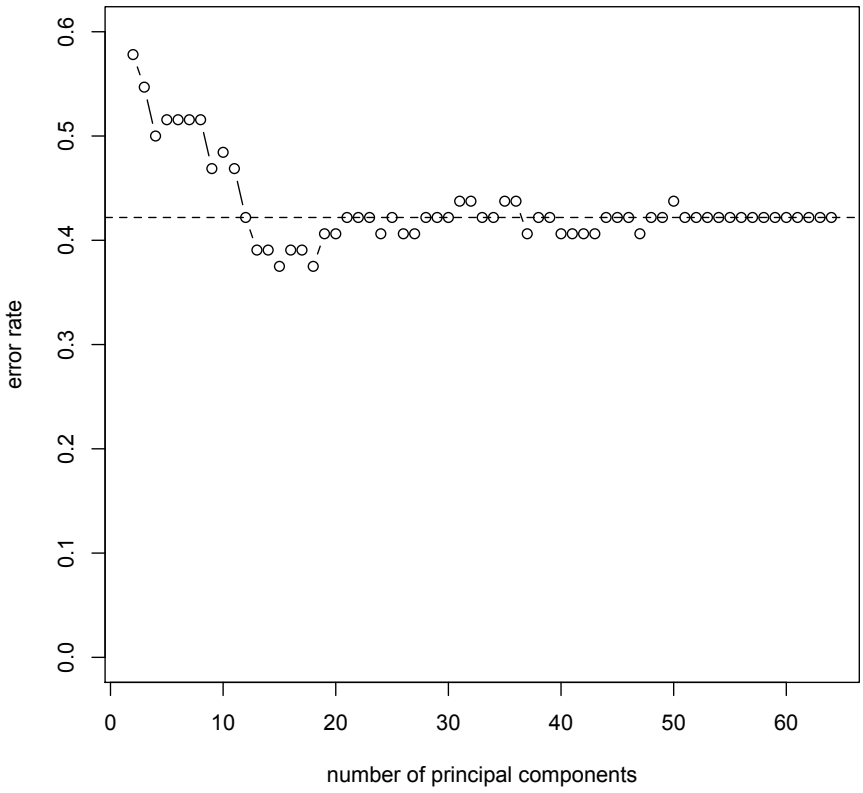

Figure 7: Solution to extra credit part of problem 7, using only unscaled principal components.

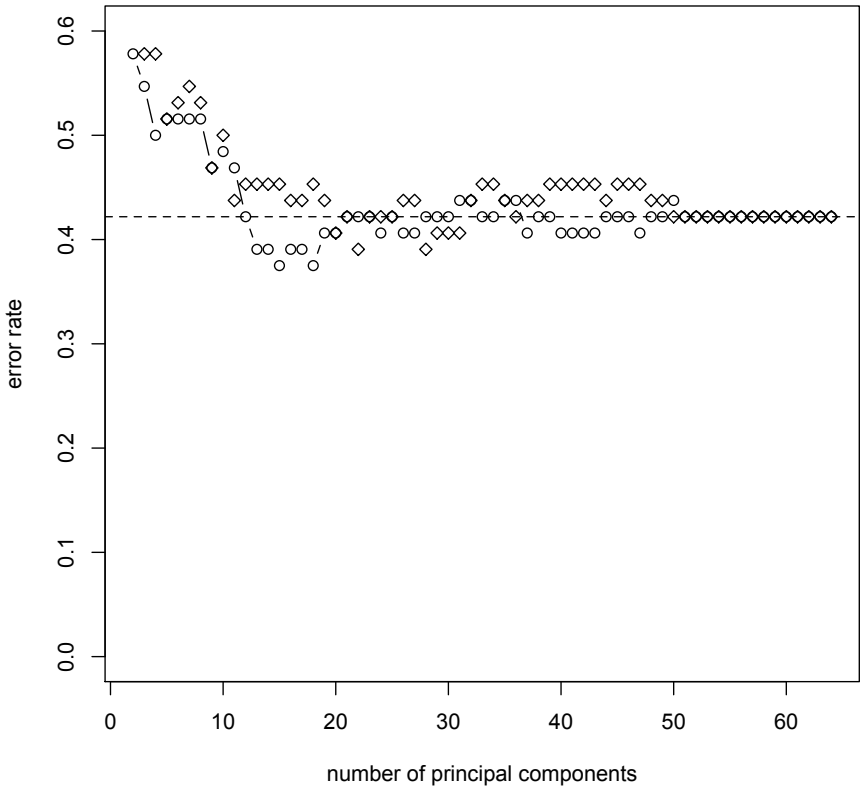

Figure 8: Solution to extra credit part of problem 7, using both unscaled and scaled principal components (circles and diamonds, respectively).