Final Project Report

Vivek Kantamani (vk2389)

Data Set

We are given a data set collected from a study of breast cancer.
The original data set contains expression levels of 24,187 genes for 97 patients.
46 are classified as relapse (`status = 1`), and 51 are classified as non-relapse (`status = 0`).

The training data set consists of 78 cases, 34 relapse (`status = 1`), and 44 non-relapse (`status = 0`).
The training data set has been preprocessed - the gene expression levels were normalized and filtered by a p-value criterion resulting in a selection of 4918 genes.
Only the training data set of dimension 78×4918 was provided.

Methodology - Kernel SVM Model, 10 x 10-fold CV

The binary classification problem that we have been given is a high-dimensional problem, $p = 4918$.
I selected the SVM model, because the SVM algorithm is rather insensitive to a large dimension p.
The SVM model also supports a variety of non-linear kernel approaches to accomodate classes that are not linearly separable, and many regularization options (L1, L2, SCAD - smoothly clipped absolute deviation, elastic SCAD, etc.) for SVM are well-supported using packages.

The binary classification problem that we have been given also has a small number of observations, $n = 78$.
In k-fold cross-validation, for a small training set, we typically choose a large value of k to mitigate the impact of error.
However, we must also consider the bias-variance trade-off when selecting an ideal value of k.
I selected 10-fold cross-validation to validate my model so as to not suffer from neither excessively high bias nor high variance in my estimates.
The 10-fold cross-validation procedure was averaged over 10 repetitions, for 10 x 10-fold cross-validation.

Read in Data

First, we read in the data.

```
# Read in data
cancer_train = read.csv("breast_cancer_train.csv")

# Convert "status" to a factor variable
cancer_train$status = factor(cancer_train$status,
                             levels = c(0,1))

# Verify dimensions of dataset
dim(cancer_train)
```

```
## [1]   78 4919
```

```
dim(cancer_train[cancer_train$status==0,])[1]
```

```
## [1] 44
```

```
dim(cancer_train[cancer_train$status==1,])[1]
```

```
## [1] 34
```

Linear SVM

Although it is extremely unlikely that binary classification on such high dimensional data could be achieved using a linear hyperplane, we proceed with linear SVM for a baseline comparison of non-linear kernel SVM methods.

The `e1071` library is used to perform SVM.
The `tune()` function in this library allows us to perform k-fold cross-validation over the model parameters.
Using the `tunecontrol` argument, we select $k = 10$ for 10-fold cross validation.
For brevity, a table of the 10 x 10-fold CV misclassification error for various values of the cost parameter is given below.

Cost	$1e-4$	$1e-3$	$1e-2$	$1e-1$	$1e+0$	$1e+1$	$1e+2$	$1e+3$	$1e+4$
10 x 10-fold CV Error	0.4321	0.3829	0.4321	0.4321	0.4321	0.4321	0.4321	0.4321	0.4321

Note that the 10 x 10-fold CV errors are approximately 40%.
The spread of these errors between each repitition of 10-fold CV is also quite large, depending on the observations that fall in each fold.
Ideally, we would like to achieve a test error less than 40%.

The code that was utilized is given below.

```
library(e1071)

tune.out_lin = tune(svm,
                    status ~ .,
                    data = cancer_train,
                    kernel = "linear",
                    tunecontrol = tune.control(cross = 10),
                    ranges = list(cost=c(1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4)))

summary(tune.out_lin)
```

Non-Linear Polynomial Kernel SVM

We now proceed with non-linear kernel SVM.
The `e1071` library is used to perform polynomial kernel SVM.
The `tune()` function in this library allows us to perform k-fold cross-validation over the model parameters.
Using the `tunecontrol` argument, we select $k = 10$ for 10-fold cross validation due to the small size of the training set.
For brevity, a table of the 10 x 10-fold CV misclassification error for various values of the polynomial kernel parameters is given below.

Cost	Degree	Gamma	Coef0	10 x 10-fold CV Error
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	1	1	0	0.4152
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	2	1	0	0.3875
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	1	2	0	0.4223
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	2	2	0	0.3902
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	1	1	1	0.4304
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	2	1	1	0.3765
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	1	2	1	0.4204
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	2	2	1	0.3887
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	1	1	2	0.4204
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	2	1	2	0.3765
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	1	2	2	0.4204
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3$	2	2	2	0.3902

Note that the 10 x 10-fold CV errors are approximately 40%.
The spread of these errors between each repitition of 10-fold CV is also quite large, depending on the observations that fall in each fold.
Ideally, we would like to achieve a test error less than 40%.

The code that was utilized is given below.

```
library(e1071)
```

```
tune.out_poly = tune(svm,
                     status ~ .,
                     data = cancer_train,
                     kernel = "polynomial",
                     ranges = list(cost=c(1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1e+0, 1e+1, 1e+2, 1e3, 1e4),
                                   degree = c(1, 2, 3, 4),
                                   gamma = c(0, 1, 2),
                                   coef0 = c(0, 1, 2)))

summary(tune.out_poly)
tune.out_poly$bestmodel
```

Non-Linear RBF Kernel SVM

The `e1071` library is used to perform RBF kernel SVM.
The `tune()` function in this library allows us to perform k-fold cross-validation over the model parameters.
Using the `tunecontrol` argument, we select $k = 10$ for 10-fold cross validation due to the small size of the training set.
For brevity, a table of the 10 x 10-fold CV misclassification error for various values of the RBF kernel parameters is given below.

Cost	Gamma	10 x 10-fold CV Error
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3,\ 1e+4$	$1e-3$	0.4339
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3,\ 1e+4$	$1e-2$	0.4339
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3,\ 1e+4$	$1e-1$	0.4339
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3,\ 1e+4$	$1e+0$	0.4339
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3,\ 1e+4$	$1e+1$	0.4339
$1e-3,\ 1e-2,\ 1e-1,\ 1e+0,\ 1e+1,\ 1e+2,\ 1e+3,\ 1e+4$	$1e+2$	0.4339

Note that the 10 x 10-fold CV errors are consistently greater than 40%.
Interestingly, there is a significant amount of stability in the 10 x 10 fold CV error values over a very large range of cost and gamma parameter values for RBF kernel SVM.
Ideally, we would like to achieve a test error less than 40%.

The code that was utilized is given below.

```
library(e1071)

tune.out_rad = tune(svm,
                    status ~ .,
                    data = cancer_train,
                    kernel = "radial",
                    ranges = list(cost=c(1e-3, 1e-2, 1e-1, 1e+0, 1e+1, 1e+2, 1e+3, 1e+4),
                                  gamma = c(1e-3, 1e-2, 1e-1, 1e+0, 5e+0, 1e+1, 1e+2)))

summary(tune.out_rad)
tune.out_rad$bestmodel
```

SVM with Recursive Feature Elimination (RFE)

In our previous attempts, we note that the estimated test misclassification error remained at or above our target of 40%.
Due to the high-dimensional nature of our data, in order to improve the fidelity of our model, we proceed with dimensionality reduction techniques.

A common method for feature selection in conjunction with SVM is recursive feature elimination (RFE).
A popular library in R, `sigFeature`, implements RFE (selecing features using the t-statistic) in conjunction with SVM in the `e1071` library.

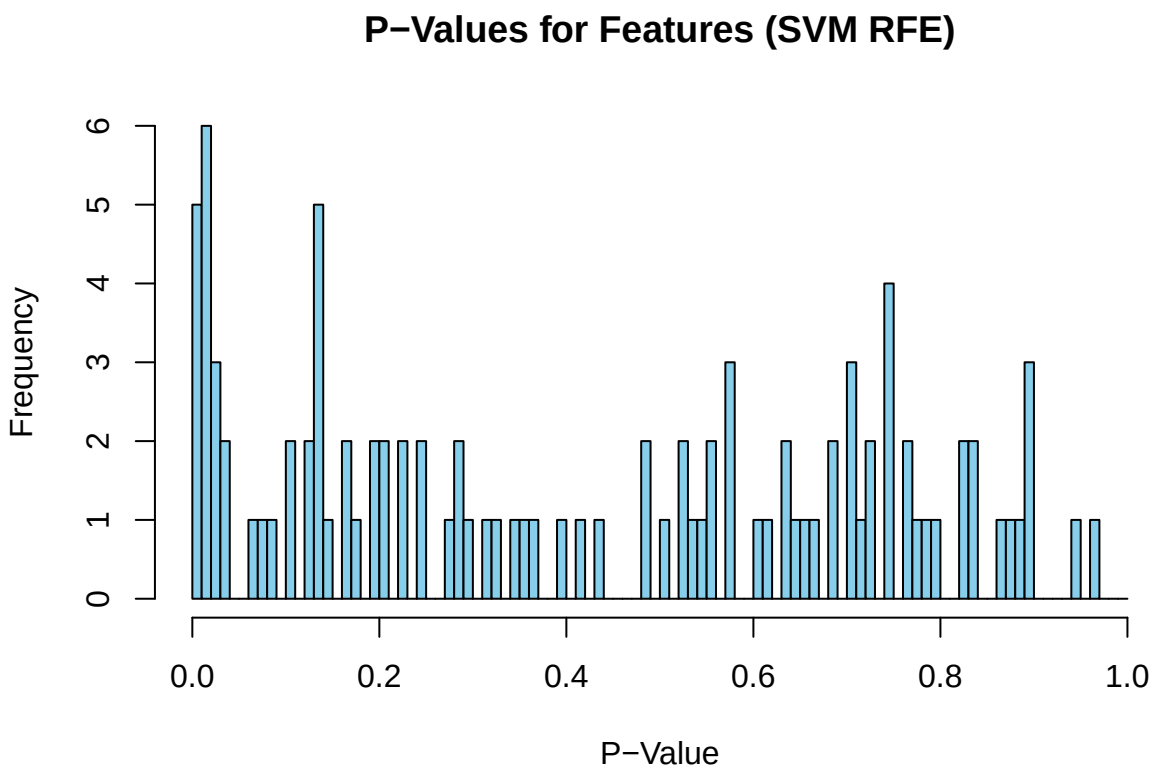
The `sigFeature` library provides two different options for recursive feature elimination.
Using the function `svmrfeFeatureRanking()`, a SVM linear kernel model is trained and RFE is performed by eliminating the feature containing the smallest ranking.
Using the function `sigFeature()`, a SVM model is trained, features are ranked using a t-statistic, and RFE is performed by eliminating the feature containing the least significant t-statistic.

We compare the utility of both feature ranking functions on our data set by comparing the p-values of the ranked features between the two classes.

```
library(sigFeature)

# Extract observations and true labels for data separately
x_data = cancer_train[,-1]
y_labels = ifelse(cancer_train$status == 0, -1, 1)
y_labels = factor(y_labels,
                  levels = c(-1,1))

# Name the rows for each observation
row_label = rep("Obs",dim(cancer_train)[1])

for(i in 1:dim(x_data)[1]){
  row_label[i] = paste(row_label[i],
                       i,
                       sep = "")
}

row.names(x_data) = row_label

# Perform RFE using svmrfeFeatureRanking()
data_sigfeat = svmrfeFeatureRanking(x_data,y_labels)
featureRankedList[1:10]
```

```
##  [1] 1073 1404 1152    5 1253 1557  105 1207  792   57
```

```
# Extract p-values for svmrfeFeatureRanking() and produce a histogram
pvals = sigFeaturePvalue(x_data, y_labels, 100, featureRankedList)

hist(unlist(pvals),
     xlab = "P-Value",
     ylab = "Frequency",
     main = "P-Values for Features (SVM RFE)",
     breaks = seq(0, 1.0, 0.01),
     col = "skyblue")
```

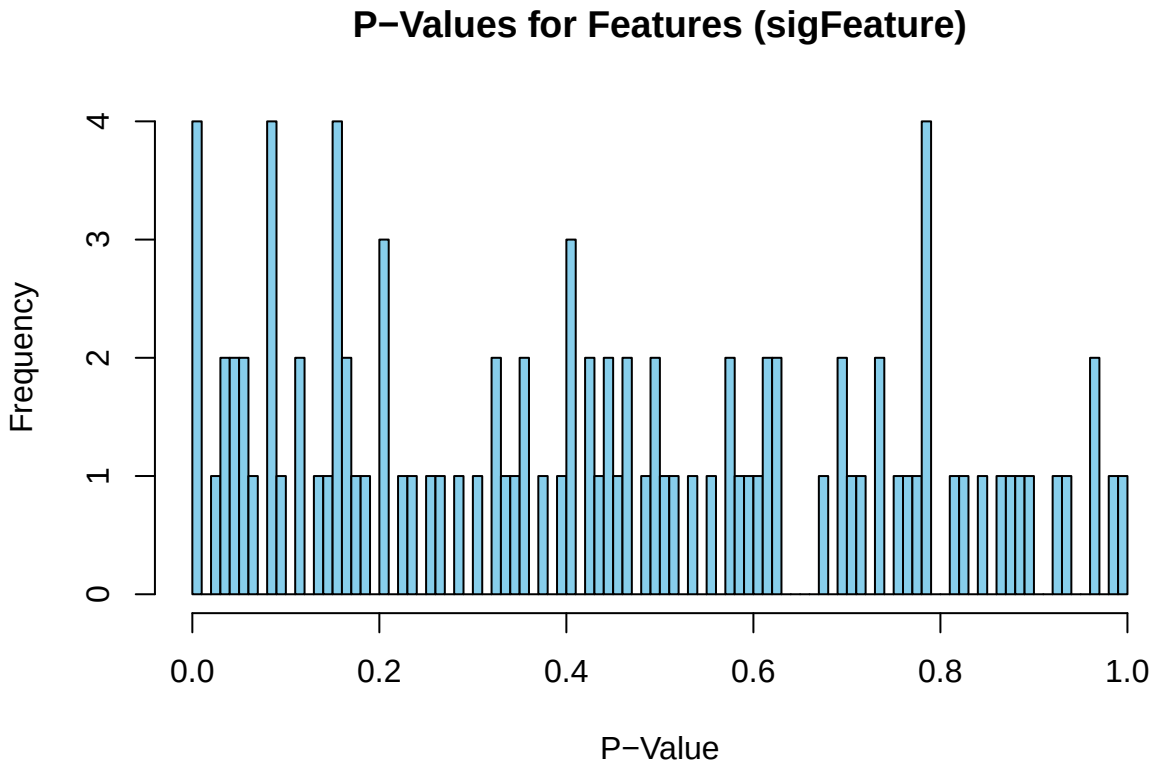



```
library(sigFeature)

# Extract observations and true labels for data separately
x_data = cancer_train[,-1]
y_labels = ifelse(cancer_train$status == 0, -1, 1)
y_labels = factor(y_labels,
                  levels = c(-1,1))

# Name the rows for each observation
row_label = rep("Obs",dim(cancer_train)[1])

for(i in 1:dim(x_data)[1]){
  row_label[i] = paste(row_label[i],
                       i,
                       sep = "")
}

row.names(x_data) = row_label

# Perform RFE using sigFeature()
data_sigfeat2 = sigFeature(x_data,y_labels)
sigfeatureRankedList[1:10]
```

```
##  [1] 2064  370 2032 2035 1519 1573 1446 2105  997  611
```

```
# Extract p-values for sigFeature() and produce a histogram
pvals2 = sigFeaturePvalue(x_data, y_labels, 100, sigfeatureRankedList)

hist(unlist(pvals2),
     xlab = "P-Value",
     ylab = "Frequency",
     main = "P-Values for Features (sigFeature)",
     breaks = seq(0, 1.0, 0.01),
```

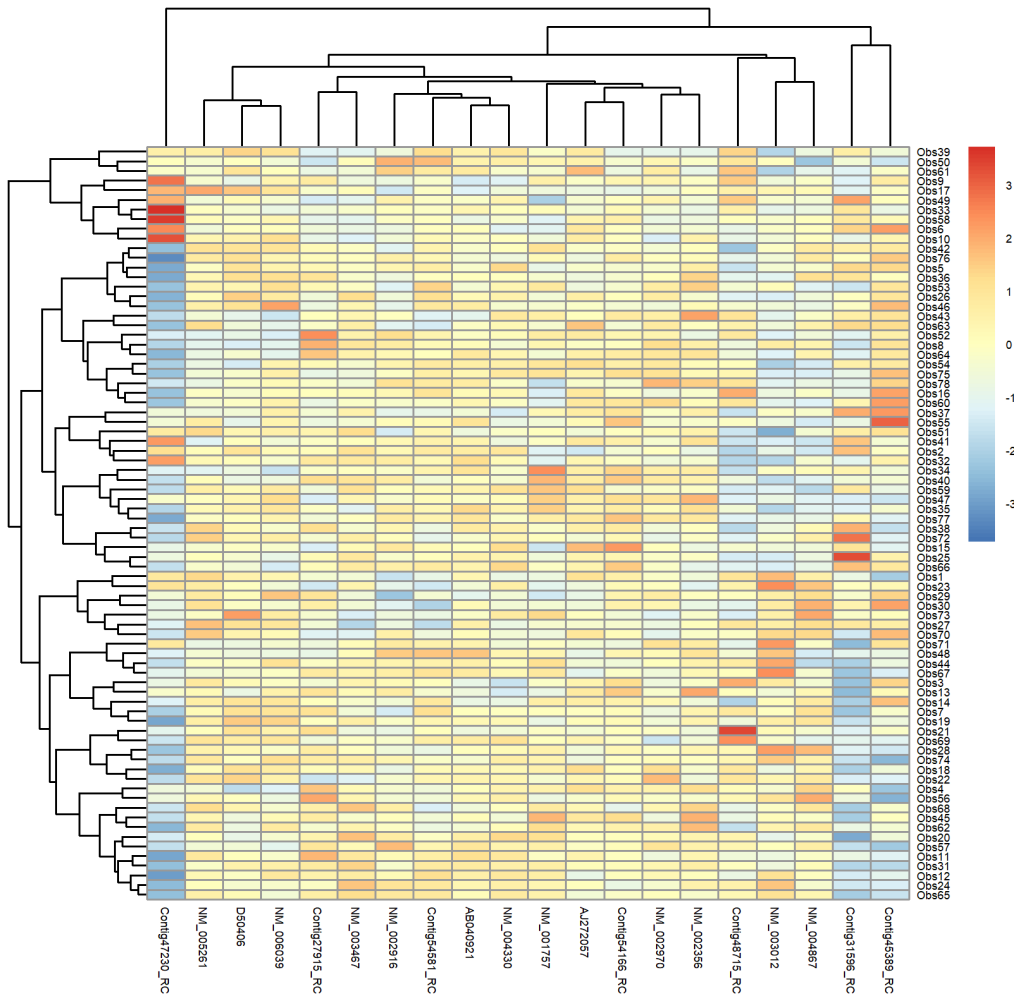
```
     col = "skyblue")
```



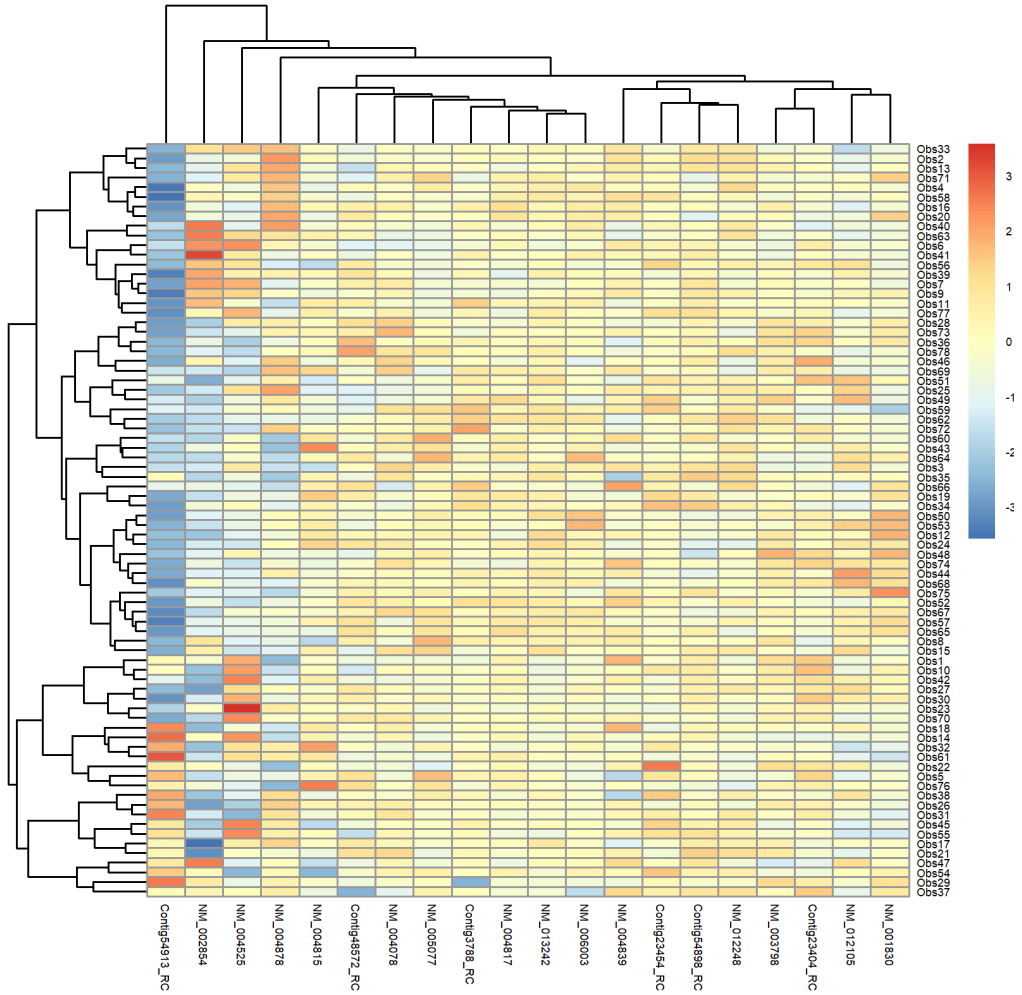
It is likely that a small number of features (relative to $p = 4918$) will be selected in our final model.
We note that the `svmrfeFeatureRanking()` function has a greater number of features below an arbitrary p-value thresholds of 0.05, 0.10, 0.20, etc. compared to the `sigFeature()` function.

We can additionally visualize the importance of the 20 most significant features obtained via each method for each of our 78 observations using heatmaps.

```
library(pheatmap)
library(RColorBrewer)

# Select 20 most significant features for svmrfeFeatureRanking() and produce a heatmap
selected_features = featureRankedList[1:20]

png("Heatmap 1.png", units="px", width=1600, height=1600, res=300)
pheatmap(x_data[,selected_features],
         scale = "row",
         cex = 0.65,
         clustering_distance_rows = "correlation")
dev.off()
```




```
library(pheatmap)
library(RColorBrewer)

# Select 20 most significant features for sigFeature() and produce a heatmap
selected_features = sigfeatureRankedList[1:20]

png("Heatmap 2.png", units="px", width=1600, height=1600, res=300)
pheatmap(x_data[,selected_features],
         scale = "row",
         cex = 0.65,
         clustering_distance_rows = "correlation")
dev.off()
```



We proceed with SVM RFE using the `svmrfeFeatureRanking()` function.

Non-Linear Polynomial Kernel SVM with RFE

Per our conclusions from performing SVM without RFE, we proceed with non-linear kernel SVM with RFE using the `svmrfeFeatureRanking()` function.

The `e1071` library is used to perform polynomial kernel SVM after RFE.
The `tune()` function in this library allows us to perform k-fold cross-validation over the model parameters.
Using the `tunecontrol` argument, we select $k = 10$ for 10-fold cross validation due to the small size of the training set.
For brevity, a table of the 10 x 10-fold CV misclassification error for the optimal values of the polynomial kernel parameters and RFE is given below.

	RFE (Number of Features)	Cost	Degree	Gamma	Coef0	10 x 10-fold CV Error
Model 1	20	0.01	2	0.5	2	0.3300
Model 2	20	0.01	2	0.3	4	0.2988
Model 3	20	0.01	2	0.4	4	0.3013
Model 4	20	0.01	2	0.3	5	0.2989
Model 5	20	0.01	2	0.4	5	0.3000

The code that was utilized is given below.

```
library(e1071)

selected_features = featureRankedList[1:20]
selected_features = c(selected_features, 1)
reduced_cancer_train = cancer_train[,selected_features]
```

```
reduced_cancer_train$status = factor(reduced_cancer_train$status,
                                     levels = c(0,1))

tune.out_polyrfe = tune(svm,
                        status ~ .,
                        data = reduced_cancer_train,
                        tunecontrol = tune.control(cross = 10),
                        type = "C-classification",
                        kernel = "polynomial",
                        ranges = list(cost = c(1e-2),
                                      degree = c(2),
                                      gamma = c(0.3, 0.4, 0.5),
                                      coef0=c(2, 3, 4, 5)))

summary(tune.out_polyrfe)
tune.out_polyrfe$bestmodel
```

```
mod1 = c(0.3357143, 0.3160714, 0.3428571, 0.3178571, 0.3214286,
              0.3446429, 0.3232143, 0.2928571, 0.3500000, 0.3553571)
summary(mod1)
sd(mod1)

mod2 = c(0.3107143, 0.3053571, 0.2910714, 0.2928571, 0.2964286,
              0.3071429, 0.2982143, 0.2946429, 0.3000000, 0.2910714)
summary(mod2)
sd(mod2)

mod3 = c(0.3107143, 0.3035714, 0.3053571, 0.2803571, 0.2964286,
              0.3071429, 0.2982143, 0.2946429, 0.3125000, 0.3035714)
summary(mod3)
sd(mod3)

mod4 = c(0.3107143, 0.2928571, 0.2928571, 0.2928571, 0.3214286,
              0.2946429, 0.2982143, 0.2946429, 0.3000000, 0.2910714)
summary(mod4)
sd(mod4)

mod5 = c(0.3107143, 0.2910714, 0.2928571, 0.2928571, 0.3089286,
              0.2946429, 0.2982143, 0.3071429, 0.3125000, 0.2910714)
summary(mod5)
sd(mod5)
```

Non-Linear RBF Kernel SVM with RFE

The `e1071` library is used to perform RBF kernel SVM after RFE.
The `tune()` function in this library allows us to perform k-fold cross-validation over the model parameters.
Using the `tunecontrol` argument, we select $k = 10$ for 10-fold cross validation due to the small size of the training set.
For brevity, a table of the 10 x 10-fold CV misclassification error for the optimal values of the RBF kernel parameters and RFE is given below.

	RFE (Number of Features)	Cost	Gamma	10 x 10-fold CV Error
Model 6	30	$7e+0$	$7e-3$	0.3174
Model 7	30	$1e+1$	$7e-3$	0.3144
Model 8	30	$5e+0$	$1e-2$	0.3170
Model 9	30	$7e+0$	$1e-2$	0.3295

The code that was utilized is given below.

```
library(e1071)
```

```
selected_features = featureRankedList[1:30]
selected_features = c(selected_features, 1)
reduced_cancer_train = cancer_train[,selected_features]

reduced_cancer_train$status = factor(reduced_cancer_train$status,
                                     levels = c(0,1))

tune.out_rbfrfe = tune(svm,
                       status ~ .,
                       data = reduced_cancer_train,
                       tunecontrol = tune.control(cross = 10),
                       type = "C-classification",
                       kernel = "radial",
                       ranges = list(cost=c(5e+0, 7e+0, 1e+1),
                                     gamma = c(5e-3, 7e-3, 1e-2)))

summary(tune.out_rbfrfe)
tune.out_rbfrfe$bestmodel
```

```
mod6 = c(0.2839286, 0.3125000, 0.2928571, 0.2928571, 0.3375000,
             0.3428571, 0.2946429, 0.3071429, 0.3321429, 0.3339286)
summary(mod6)
sd(mod6)

mod7 = c(0.2875000, 0.3214286, 0.2660714, 0.3178571, 0.3089286,
             0.3303571, 0.2946429, 0.3196429, 0.3571429, 0.3464286)

summary(mod7)
sd(mod7)

mod8 = c(0.2857143, 0.3339286, 0.2928571, 0.3053571, 0.3250000,
             0.3178571, 0.3071429, 0.3214286, 0.3571429, 0.3339286)
summary(mod8)
sd(mod8)

mod9 = c(0.3125000, 0.3482143, 0.3035714, 0.3178571, 0.3089286,
             0.3178571, 0.3214286, 0.3571429, 0.3696429, 0.3357143)
summary(mod9)
sd(mod9)
```

Final Model

Per the results of model selection above, clearly Model 2 (polynomial kernel SVM after RFE) yielded the best results.

	RFE (Number of Features)	Cost	Degree	Gamma	Coef0	10 x 10-fold CV Error
Model 2	20	0.01	2	0.3	4	0.2988

It demonstrates the lowest 10 x 10-fold CV error of 0.2988 and the smallest standard deviation of 0.006944.

```
# Summary statistics for the 10-fold CV errors over 10 repitions (10 x 10-fold CV error)
mod2 = c(0.3107143, 0.3053571, 0.2910714, 0.2928571, 0.2964286,
         0.3071429, 0.2982143, 0.2946429, 0.3000000, 0.2910714)

summary(mod2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2911  0.2933  0.2973  0.2988  0.3040  0.3107
```

```
sd(mod2)
```

```
## [1] 0.006944172
```

We now export and save the final model.

```
library(e1071)

# Select the 20 most significant features otained using RFE
selected_features = featureRankedList[1:20]
selected_features = c(selected_features, 1)

# Produce a data set containing only the selected features
reduced_cancer_train = cancer_train[,selected_features]
reduced_cancer_train$status = factor(reduced_cancer_train$status,
                                     levels = c(0,1))

# We demonstrate the 10-fold CV error one final time prior to finalizing our model
model2 = tune(svm,
              status ~ .,
              data = reduced_cancer_train,
              tunecontrol = tune.control(cross = 10),
              type = "C-classification",
              kernel = "polynomial",
              ranges = list(cost = c(1e-2),
                            degree = c(2),
                            gamma = c(0.3),
                            coef0=c(4)))

summary(model2)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.2964286
```

```
# Construct the final model using the parameters of Model 2
final_model = svm(status ~ .,
                  data = reduced_cancer_train,
                  type = "C-classification",
                  kernel = "polynomial",
                  cost = 1e-2,
                  degree = 2,
                  gamma = 0.3,
                  coef0 = 4)

summary(final_model)
```

```
##
## Call:
## svm(formula = status ~ ., data = reduced_cancer_train, type = "C-classification",
##     kernel = "polynomial", cost = 0.01, degree = 2, gamma = 0.3,
##     coef0 = 4)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  4
##
## Number of Support Vectors:  67
##
##  ( 35 32 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  0 1
```

```
# Use saveRDS() to extract the model object
saveRDS(final_model,
        file = "final_model.RDS")
```

We now construct our function which wraps our final model, takes gene expression levels as input, and returns the prediction of patients' status.

```
# Function argument is the test data set
model_function = function(test_file) {

  # Read in test data set
  test_data = read.csv(test_file)

  # Load final model using readRDS()
  final_model = readRDS(file = "final_model.RDS")

  # Predict the class labels for the test data set
  pred = predict(final_model, test_data)

  # Return the prediction of patients' status
  return(pred)
}
```

The function is provided in the `Model Evalution.RMD` file in order to evaluate the model using the test set.