STA 521 HW5

William Tirone

The Honor Code:

> **!** Important
>
> (a) Please state the names of people who you worked with for this homework. You can also provide your comments about the homework here.
> *Eli Gnesin, Natalie Smith, Alonso Guererro*
> (b) Please type/write the following sentences yourself and sign at the end. We want to make it extra clear that nobody cheats even unintentionally.
> *I hereby state that all of my solutions were entirely in my words and were written by me. I have not looked at another student's solutions and I have fairly credited all external sources in this write up.*

1 True or False

1. FALSE, it can be kernelized as we saw in Lecture 19
2. FALSE, from ISL figure 9.4, the hard margin SVM does not work if the data is not linearly separable.
3. TRUE, that is how the margins are defined mathematically.
4. FALSE, as we increase C, the soft SVM will approach the hard margin formulation which allows for no points. So we'll have fewer points as it increases.

5. FALSE, as C increases it approaches the hard margin.
6. FALSE, it will be at least as large if not greater because of the additional constraint.
7. TRUE, referencing wiki for a Gram matrix.
8. FALSE, function must involve only inner products.

2)

handwritten

3)

handwritten

4)

1-3 handwritten

4.4)

My implementation of kernel PCA. I wrote a separate function to build the kernel matrix for a polynomial kernel.

```
poly_kernel = function(X,p=2,c=1){
  # takes design matrix X as input and outputs
  # Kernel matrix K

  # initialize matrix K with dimension n x n
  n = dim(X)[1]
  K = matrix(NA,n,n)
```

```
  # compute matrix element-wise
  for (i in 1:n){
    for (j in 1:n){
      K[i,j] = (t(X[i,]) %*% X[j,] + c)^p
    }
  }
  return(K)
}


mykpca = function(X,kernel='poly'){

  X = as.matrix(X)

  # kernel matrix
  if (kernel=='poly'){
  K = poly_kernel(X) }
  else {print("kernel not supported")}

  # helmert matrix
  n = dim(K)[1]
  H = diag(n) - (1/n) * matrix(1,n,n)

  # output matrix
  output = matrix(NA,nrow=dim(X)[1],ncol=2)

  # normalized kernel matrix
  K.tilde = H %*% K %*% H

  # find eigs of normalized kernel matrix
  eigs = eigen(K.tilde)

  # compute projects to the 1st and 2nd PCs
  e1 = eigs$vectors[,1]
  e2 = eigs$vectors[,2]
  for (q in 1:n){
    x = X[q,]
    a = 0
    b = 0
    for(z in 1:n){
      a = a + e1[z] * K[q,z]
```

```
      b = b + e2[z] * K[q,z]
    }
    output[q,] = c(a,b)
  }
  return(output)
}
```

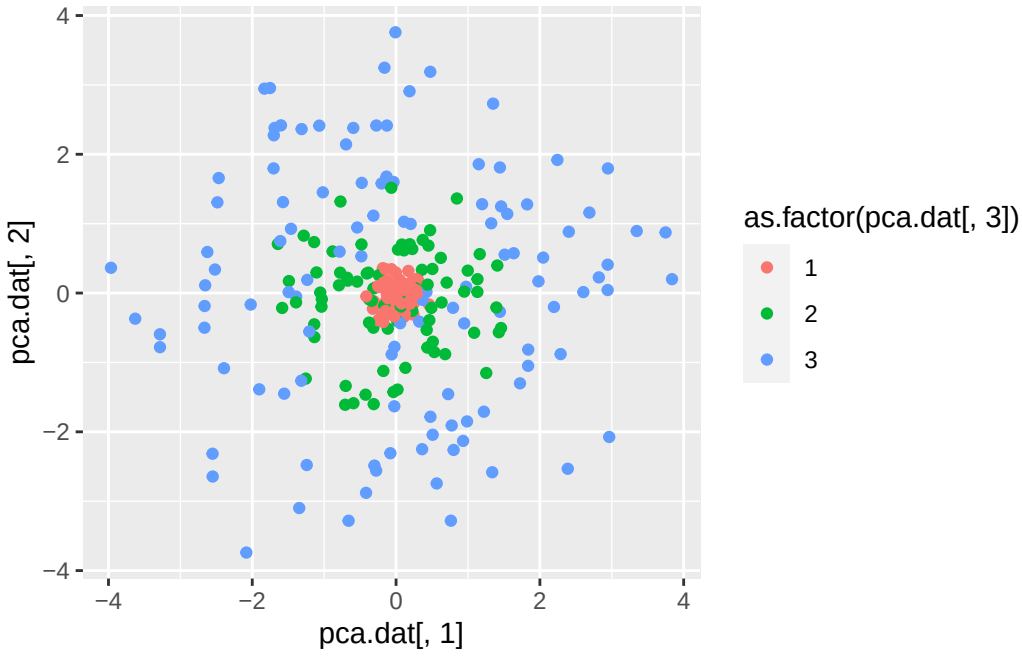
4.5)

```
# generate data
# setup from HW
set.seed(123456)
n = 300
p = 10
K = 3
xr = matrix(rnorm(n*p),nrow=n)
y = sample(1:K, n, replace=T)
x = diag(y^2) %*% t(apply(xr, 1, function(x)(x/sqrt(sum(x^2))))) +
0.001 * matrix(rnorm(n*p),nrow=n)
dfX = data.frame(x)
```

Performing regular PCA on dfX, the plot below is not linearly separable, so we can try to use the kernelized version of PCA to turn them into a linearly separable data set

```
pca.model = prcomp(dfX,center=TRUE, scale=TRUE)
pca.dat = data.frame(cbind(pca.model$x[,1:2], y))

ggplot() +
  geom_point(data=pca.dat, aes(x=pca.dat[,1],y=pca.dat[,2], color = as.factor(pca.dat[,3])
```

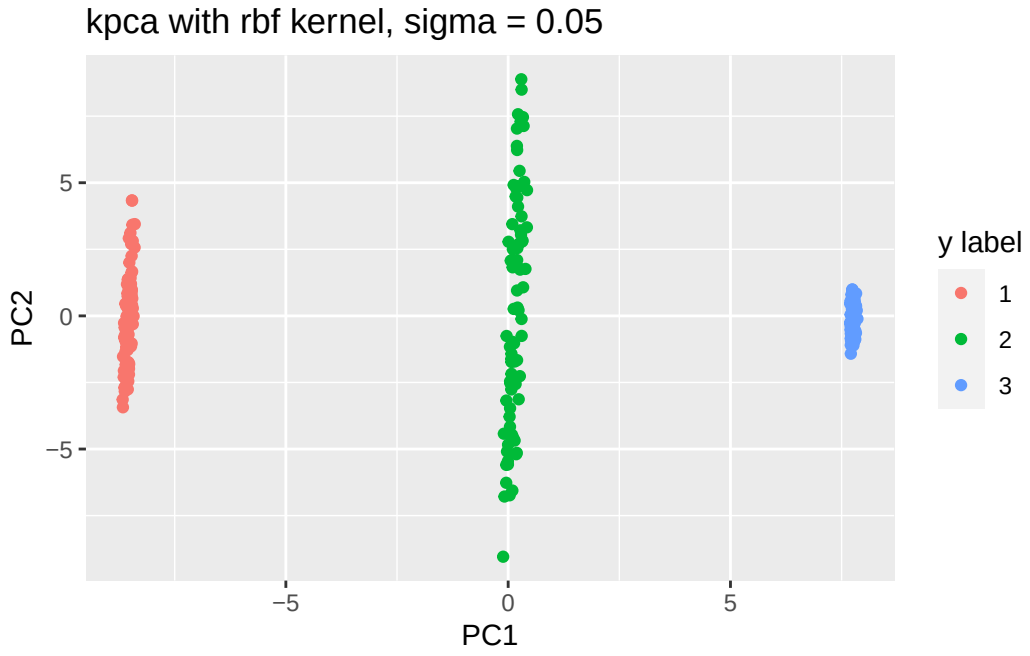


4.6)

Now we can separate the points with linear classifiers easily on the kernel PCA plot using the radial basis kernel.

```
pca = kpca(~., dfX, kernel='rbfdot', kpar=list(sigma=0.05))
part.6.data = data.frame(cbind(pca@rotated[,1:2],as.factor(y)))

ggplot() +
  geom_point(data=part.6.data,
             aes(x=part.6.data[,1],
                 y=part.6.data[,2],
                 color = as.factor(part.6.data[,3]))) +
  labs(title="kpca with rbf kernel, sigma = 0.05",
       x = "PC1",
       y = "PC2",
       color="y label")
```

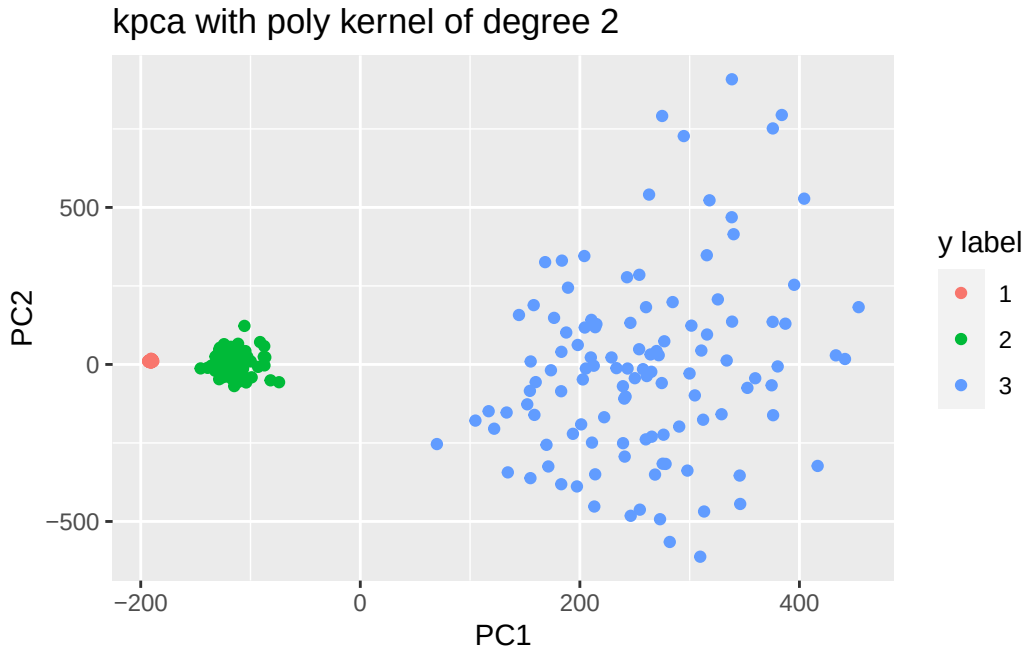


4.7)

Repeating the above but with a polynomial kernel.

```
pca = kpca(~., dfX, kernel='polydot', kpar=list(degree=2,offset=1))
part.7.data = data.frame(cbind(pca@rotated[,1:2],as.factor(y)))

ggplot() +
  geom_point(data=part.7.data,
             aes(x=part.7.data[,1],
                 y=part.7.data[,2],
                 color = as.factor(part.7.data[,3]))) +
  labs(title="kpca with poly kernel of degree 2",
       x = "PC1",
       y = "PC2",
       color="y label")
```

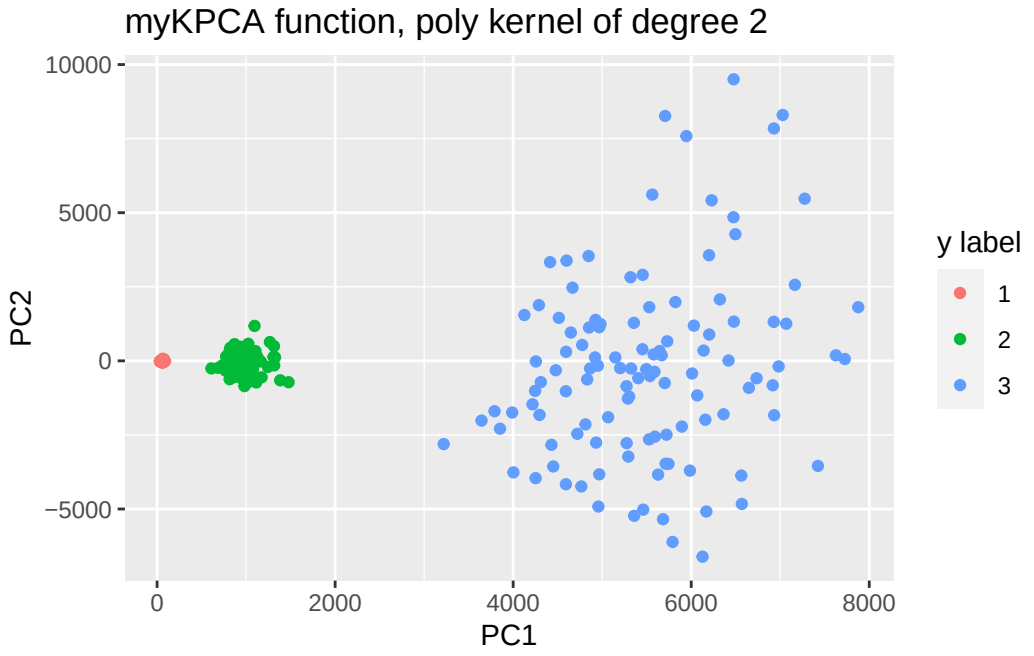


4.8)

Using `mykpca` function below, the results look identical to the output in question 7 from r's implementation of `kpca` . We can see the data is linearly separable now which accomplishes the goal of kernelized PCA.

```
my_data = mykpca(dfX)
my_data = data.frame(cbind(my_data,as.factor(y)))

ggplot() +
  geom_point(data=my_data,
             aes(x=my_data[,1],
                 y=my_data[,2],
                 color = as.factor(my_data[,3]))) +
  labs(title="myKPCA function, poly kernel of degree 2",
     x = "PC1",
     y = "PC2",
     color="y label")
```


myKPCA function, poly kernel of degree 2
10000
5000
0
−5000
PC2
0
2000
4000
6000
8000
PC1
y label
1
2
3