Assignment_3_Report

Jakub Domasik

7 12 2019

Q1

(a)

Here is the provided gradient descent function.

```r
gradient.descent <- function(f, gradf, x0, iterations=1000, eta=0.2) {
x<-x0
for (i in 1:iterations) {
  if(i<50) {
cat(i,"/",iterations,": ",x," ",f(x),"\n")
  }
x<-x-eta*gradf(x)
}
x
}
```

```r
gradient.ascent <- function(f, df, x0, iterations=1000, eta=0.2) {
  gradient.descent(f, df, x0, iterations, -eta)
}
```

To test the function, the following code is used.

```r
f <-function(x) { (1+x^2)^(-1) }
gradf<-function(x) { -2*x*(1+x^2)^(-2) }
gradient.ascent(f,gradf,3,40,0.5)
```

```
## 1 / 40 :  3   0.1
## 2 / 40 :  2.97   0.1018237
## 3 / 40 :  2.939207   0.1037459
## 4 / 40 :  2.907572   0.1057756
## 5 / 40 :  2.87504   0.1079231
## 6 / 40 :  2.841554   0.1101998
## 7 / 40 :  2.807046   0.1126189
## 8 / 40 :  2.771444   0.1151954
## 9 / 40 :  2.734667   0.1179467
## 10 / 40 :  2.696624   0.120893
## 11 / 40 :  2.657212   0.1240575
## 12 / 40 :  2.616317   0.1274679
## 13 / 40 :  2.573807   0.1311564
## 14 / 40 :  2.529532   0.1351619
## 15 / 40 :  2.483321   0.1395307
## 16 / 40 :  2.434974   0.144319
## 17 / 40 :  2.384258   0.1495956
## 18 / 40 :  2.330901   0.155446
```

```
## 19 / 40 :  2.274579   0.1619772
## 20 / 40 :  2.214901   0.1693254
## 21 / 40 :  2.151398   0.1776669
## 22 / 40 :  2.083488   0.1872336
## 23 / 40 :  2.010448   0.1983379
## 24 / 40 :  1.931361   0.2114095
## 25 / 40 :  1.845041   0.2270572
## 26 / 40 :  1.74992   0.2461708
## 27 / 40 :  1.643875   0.2701006
## 28 / 40 :  1.523947   0.300986
## 29 / 40 :  1.385889   0.3423852
## 30 / 40 :  1.223424   0.400518
## 31 / 40 :  1.027169   0.4866
## 32 / 40 :  0.7839563   0.6193532
## 33 / 40 :  0.4832319   0.8106927
## 34 / 40 :  0.165641   0.9732957
## 35 / 40 :  0.008728518   0.9999238
## 36 / 40 :  1.329848e-06   1
## 37 / 40 :  4.703997e-18   1
## 38 / 40 :  0   1
## 39 / 40 :  0   1
## 40 / 40 :  0   1

## [1] 0
```

(b) (i)

We shall prove that function $f(x_1, x_2) = (x_1 - 1)^2 + 100(x_1^2 - x_2)^2$ has one unique stationary point which is a minimum.

1. Find stationary points.

(a) Calculate derivatives of $f(x_1, x_2)$.

$$\frac{df}{dx_1} = 2(x_1 - 1) + 4 * 100x_1(x_1 - x_2) = 2x_1 - 2 + 400x_1^3 - 400x_1x_2$$
$$\frac{df}{dx_2} = 200x_1^2 + 200x_2$$

(b) Set the derivatives equal to 0.

$$(1)2x_1 - 2 + 400x_1^3 - 400x_1x_2 = 0$$
$$(2)200x_1^2 + 200x_2 = 0$$

(c) Solve the system of two equations.

$$(2) \implies x_2 = x_1^2$$
$$(1) \implies 2x_1 - 2 + 400x_1^3 - 400x_1^3 \implies x_1 = 1 \implies x_2 = 1$$

Therefore, we found a unique stationary point at $(1, 1)$

2. Check what kind of stationary point is $(1,1)$.

(a) Calculate 2^{nd} derivatives.

$$\frac{d^2f}{dx_1^2} = 2 + 1200x_1 + 400x_2$$
$$\frac{d^2f}{dx_2^2} = 200$$
$$\frac{d^2f}{dx_1x_2} = -400x_1$$
$$f(1,1) = 1602 > 0$$

$$det(D^2f) = f_{x_1x_1} * f_{x_2x_2} - f_{x_1x_2}^2 = 160400$$

$D^2f(1,1)$ is positive definite so (1,1) is a minimum. Therefore, we proved that the provided function f has a unique minimum.

(b)(ii)

For the function f as defined above, gradient of f is returned using *gradient_f* function

```
f <- function(x) (x[1]-1)^2 + 100*(x[1]^2-x[2])^2
gradient_f <- function(x) {
  return (c( 2 * x[1] - 2 + 400 * x[1]^3 - 400 * x[2] * x[1],
            -200 * x[1]^2 + 200* x[2]))
}
```

(b) (iii)

Gradient descent function is used to find the minimum of the function f. I am asked to start the gradient algorithm at $x_0 = (3,4)$. We set number of iterations to 30 000 and $alpha = 0.0007$. The converges to 0 as (x_1, x_2) goes to $(1,1)$.

```
gradient.descent(f, gradient_f, c(3, 4), 30000, 0.0007)
```

```
## 1 / 30000 :  3 4   2504
## 2 / 30000 :  -1.2028 4.7   1063.23
## 3 / 30000 :  -2.295366 4.244542   115.7505
## 4 / 30000 :  -1.63252 4.387925   303.7352
## 5 / 30000 :  -2.416337 4.146732   297.9424
## 6 / 30000 :  -1.266821 4.383606   777.2945
## 7 / 30000 :  -2.249305 3.994578   123.9369
## 8 / 30000 :  -1.574142 4.14365   284.0905
## 9 / 30000 :  -2.304723 3.910448   207.286
## 10 / 30000 :  -1.395805 4.10663   471.5913
## 11 / 30000 :  -2.235992 3.80446   153.3222
## 12 / 30000 :  -1.483173 3.971788   320.1597
## 13 / 30000 :  -2.215582 3.72371   150.7845
```

```
## 14 / 30000 :  -1.475892 3.889623   299.007
## 15 / 30000 :  -2.179647 3.650032   131.293
## 16 / 30000 :  -1.503357 3.804148   244.6803
## 17 / 30000 :  -2.149811 3.587979   116.777
## 18 / 30000 :  -1.523163 3.732699   205.9311
## 19 / 30000 :  -2.122115 3.534924   103.537
## 20 / 30000 :  -1.542299 3.670507   173.3432
## 21 / 30000 :  -2.096605 3.489652   91.6904
## 22 / 30000 :  -1.560345 3.616506   146.2276
## 23 / 30000 :  -2.073098 3.45105   81.13146
## 24 / 30000 :  -1.577323 3.569586   123.637
## 25 / 30000 :  -2.051421 3.418156   71.74812
## 26 / 30000 :  -1.593276 3.52878   104.7846
## 27 / 30000 :  -2.031414 3.390145   63.43223
## 28 / 30000 :  -1.608253 3.493255   89.02718
## 29 / 30000 :  -2.012933 3.366306   56.08149
## 30 / 30000 :  -1.6223 3.462289   75.83805
## 31 / 30000 :  -1.995847 3.346029   49.60019
## 32 / 30000 :  -1.635463 3.435262   64.78533
## 33 / 30000 :  -1.980039 3.328788   43.89947
## 34 / 30000 :  -1.647785 3.411636   55.51369
## 35 / 30000 :  -1.965402 3.314134   38.89742
## 36 / 30000 :  -1.65931 3.390948   47.72999
## 37 / 30000 :  -1.951838 3.301679   34.51896
## 38 / 30000 :  -1.67008 3.372798   41.19174
## 39 / 30000 :  -1.939261 3.291089   30.69561
## 40 / 30000 :  -1.680133 3.356839   35.6978
## 41 / 30000 :  -1.927591 3.282081   27.36518
## 42 / 30000 :  -1.689508 3.342774   31.08089
## 43 / 30000 :  -1.916757 3.274407   24.47142
## 44 / 30000 :  -1.698238 3.330344   27.20154
## 45 / 30000 :  -1.906694 3.267857   21.96359
## 46 / 30000 :  -1.70636 3.319325   23.9432
## 47 / 30000 :  -1.897344 3.262252   19.79607
## 48 / 30000 :  -1.713903 3.309524   21.20821
## 49 / 30000 :  -1.888653 3.257436   17.92792

## [1] 0.9998311 0.9996615
```

(c)

The function *gradient.momentum* performs the algorithm: gradient descent with momentum. We use it to find the minimum of the function f defined in part(b).

```
gradient.momentum <- function(f, gradf, x0, iterations = 1000, eta = 0.2, alpha) {
  x1 <- x0
  x2 <- x1 - eta * gradf(x1)
  cat(1, " : ", x1, ", ", f(x1), "\n")
  cat(2, " : ", x2, ", ", f(x2), "\n")
  for( i in 3:iterations) {
    x <- x2 - eta * gradf(x2) + alpha * (x2 - x1)
    x1 <- x2
    x2 <- x
```

```r
    if(i < 50) {
    cat(i, " : ", x, ", ", f(x), "\n")
  }
  }
  x

}

gradient.momentum(f, gradient_f, c(3,4), 30000, 0.0005, 0.03)
```

```
## 1  :  3 4 ,  2504
## 2  :  -0.002 4.5 ,  2026
## 3  :  -0.092858 4.065 ,  1646.614
## 4  :  -0.1698243 3.646313 ,  1309.979
## 5  :  -0.2938304 3.272005 ,  1016.522
## 6  :  -0.483466 2.942209 ,  735.7813
## 7  :  -0.7495622 2.661468 ,  443.9032
## 8  :  -1.070555 2.443083 ,  172.5067
## 9  :  -1.355815 2.306832 ,  27.50815
## 10  :  -1.489084 2.255885 ,  6.343879
## 11  :  -1.502063 2.250505 ,  6.263555
## 12  :  -1.498242 2.250913 ,  6.245036
## 13  :  -1.497482 2.250306 ,  6.243585
## 14  :  -1.497314 2.249503 ,  6.242283
## 15  :  -1.497074 2.248723 ,  6.240992
## 16  :  -1.496813 2.24795 ,  6.239702
## 17  :  -1.496554 2.247177 ,  6.238412
## 18  :  -1.496295 2.246404 ,  6.237121
## 19  :  -1.496037 2.24563 ,  6.235831
## 20  :  -1.495778 2.244856 ,  6.23454
## 21  :  -1.49552 2.244083 ,  6.233249
## 22  :  -1.495261 2.243309 ,  6.231958
## 23  :  -1.495002 2.242536 ,  6.230667
## 24  :  -1.494743 2.241762 ,  6.229376
## 25  :  -1.494484 2.240988 ,  6.228084
## 26  :  -1.494225 2.240214 ,  6.226793
## 27  :  -1.493966 2.239441 ,  6.225501
## 28  :  -1.493707 2.238667 ,  6.224209
## 29  :  -1.493448 2.237893 ,  6.222917
## 30  :  -1.493189 2.237119 ,  6.221625
## 31  :  -1.49293 2.236345 ,  6.220333
## 32  :  -1.49267 2.235572 ,  6.219041
## 33  :  -1.492411 2.234798 ,  6.217748
## 34  :  -1.492152 2.234024 ,  6.216455
## 35  :  -1.491892 2.23325 ,  6.215163
## 36  :  -1.491633 2.232476 ,  6.21387
## 37  :  -1.491373 2.231702 ,  6.212577
## 38  :  -1.491113 2.230928 ,  6.211284
## 39  :  -1.490854 2.230154 ,  6.20999
## 40  :  -1.490594 2.229379 ,  6.208697
## 41  :  -1.490334 2.228605 ,  6.207403
## 42  :  -1.490074 2.227831 ,  6.20611
## 43  :  -1.489814 2.227057 ,  6.204816
```

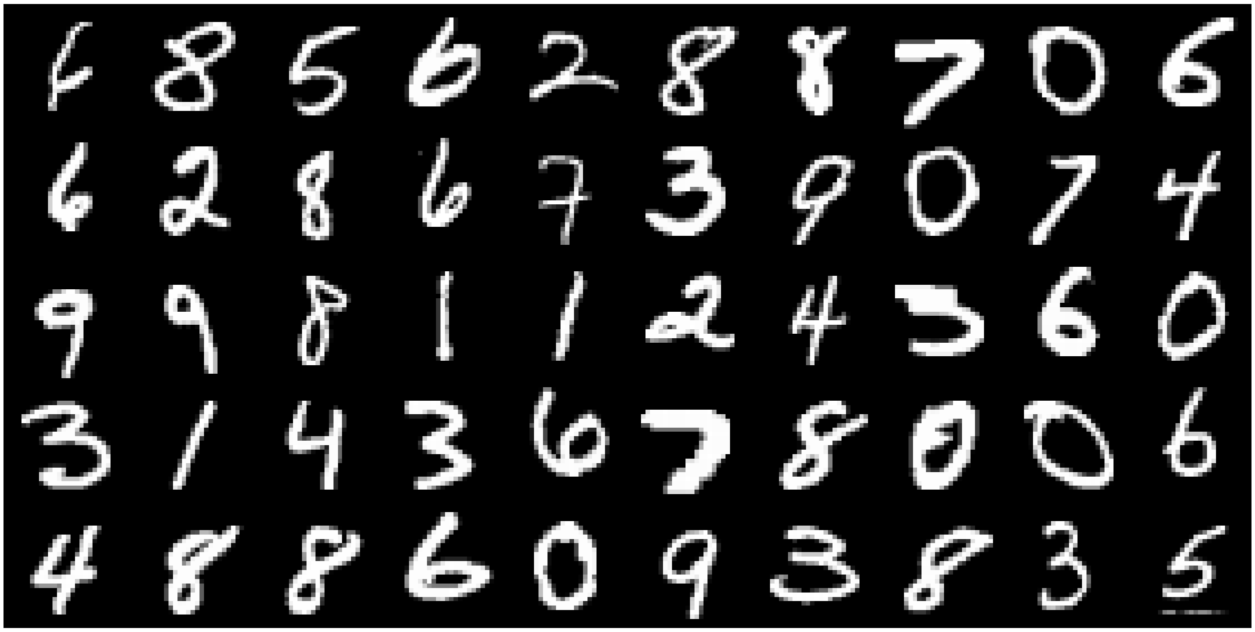
```
## 44  :  -1.489554 2.226283 ,  6.203522
## 45  :  -1.489294 2.225509 ,  6.202228
## 46  :  -1.489034 2.224734 ,  6.200933
## 47  :  -1.488774 2.22396 ,  6.199639
## 48  :  -1.488514 2.223186 ,  6.198345
## 49  :  -1.488254 2.222411 ,  6.19705

## [1] 0.9988378 0.9976723
```

Q2

(a)

```r
load("mnist.tiny.RData")
train.X=train.X/255
test.X=test.X/255
library(grid)
grid.raster(array(aperm(array(train.X[1:50,],c(5,10,28,28)),c(4,1,3,2)),c(140,280)),
            interpolate=FALSE)
```



```r
library(e1071)
```

At first, we run SVM with the linear kernel.

```r
svm(train.X,train.labels,type="C-classification",kernel="linear",cross=3)$tot.accuracy
```

```
## [1] 85.8
```

Now, we use the polynomial kernel and try different degrees to find the optimal value.

```r
svm(train.X,train.labels,type="C-classification",kernel="poly",degree=2,coef=1,cross=3)$tot.accuracy
```

```
## [1] 82
```

```r
svm(train.X,train.labels,type="C-classification",kernel="poly",degree=5,coef=1,cross=3)$tot.accuracy
```

```
## [1] 86.6
```

```r
svm(train.X,train.labels,type="C-classification",kernel="poly",degree=10,coef=1,cross=3)$tot.accuracy
```

```
## [1] 88.1
```

```r
svm(train.X,train.labels,type="C-classification",kernel="poly",degree=13,coef=1,cross=3)$tot.accuracy
```

```
## [1] 88.4
```

```r
svm(train.X,train.labels,type="C-classification",kernel="poly",degree=14,coef=1,cross=3)$tot.accuracy
```

```
## [1] 89.6
```

```r
svm(train.X,train.labels,type="C-classification",kernel="poly",degree=15,coef=1,cross=3)$tot.accuracy
```

```
## [1] 88.6
```

```r
svm(train.X,train.labels,type="C-classification",kernel="poly",degree=16,coef=1,cross=3)$tot.accuracy
```

```
## [1] 88.4
```

The SVM with linear kernel has the accuracy of 85.8% and performs better than SVM with polynomial kernel of 1st degree. SVM performs the best overall with polynomial kernel of 13th degree.

Now, we try the radial kernel for the SVM and look for the optimal gamma.

```r
set.seed(1)
svm(train.X, train.labels, type = "C-classification", kernel = "radial",
    gamma = 1, coef = 1, cross = 3)$tot.accuracy
```

```
## [1] 12.4
```

```r
set.seed(1)
svm(train.X, train.labels, type = "C-classification", kernel = "radial",
    gamma = 0.5, coef = 1, cross = 3)$tot.accuracy
```

```
## [1] 13.5
```

```r
set.seed(1)
svm(train.X, train.labels, type = "C-classification", kernel = "radial",
    gamma = 0.01, coef = 1, cross = 3)$tot.accuracy
```

```
## [1] 89
```

```r
set.seed(1)
svm(train.X, train.labels, type = "C-classification", kernel = "radial",
gamma = 0.03, coef = 1, cross = 3)$tot.accuracy
```

```
## [1] 90.9
```

```r
set.seed(1)
svm(train.X, train.labels, type = "C-classification", kernel = "radial",
    gamma = 0.02, coef = 1, cross = 3)$tot.accuracy
```

```
## [1] 90.7
```

We see that the optimal value of gamma in this case is 0.03. For this value of gamma SVM with radial kernel has higher accuracy than any of SVMs with polynomial or linear kernels.

(b)

```r
log.C.range <- log(c(0.001, 0.01, 0.1, 1, 10, 100))
log.gamma.range <- log(c(0.001, 0.01, 0.1, 1, 10, 100))

comb <- matrix(ncol = 2, nrow = length(log.gamma.range) * length(log.C.range))
count <- 0
results <- c()

for(i in 1:length(log.C.range)) {
  for(j in 1:length(log.gamma.range)) {
    count <- count + 1
    comb[count, ] <- c(gamma = exp(log.C.range[i]), exp(log.gamma.range[j]))
  }
}


for ( i in 1:nrow(comb)) {
  accuracy <- svm(train.X, train.labels, type = "C-classification", kernel = "rad",
                  cost = comb[i,1], gamma = comb[i,2], degree = 2, coef = 1, cross = 3)$tot.accuracy
  results <- c(results, accuracy)
```

```
}

row <- which.max(results)
comb[row, ]
```

```
## [1] 10.00  0.01
```

```
max(results)
```

```
## [1] 90.8
```

Ascalculated above, the optimal values for our model are: cost = 10, gamma = 0.01. Now, we train our model on the *tiny* training set and test it on the training set. In the end, we check the accuracy of our model.

```
model <- svm(train.X, train.labels, type = "C-classification", kernel = "rad",
             cost = 10, gamma = 0.01, degree = 2, coef = 1)
test.predictions <- as.vector(predict(model, test.X), mode = "numeric")

accuracy <- sum(diag(table(test.labels, test.predictions)))/sum(table(test.labels, test.predictions))

accuracy
```

```
## [1] 0.913
```

The model got the accuracy of 91.3% on the test set.