Data Mining HW4

응용통계학과 20152410 배형준

Construct support vector machine classifiers for MNIST_small data. Report the best tuning parameters and what kernel works the best. Make a confusion matrix and provide your conclusions and discussion.

caret 패키지의 train 함수를 이용하여 svmLinear, svmPoly, svmRadial 를 각각 학습하였습니다. trainControl 함수를 사용해 학습 방법을 cross validation fold=3 으로 하이퍼 파라미터를 탐색했고, 탐색 횟수는 tuneLength 인수를 이용해 10 회로 고정하였습니다. svmLinear 은 커널을 사용하지 않았기 때문에 해당 학습 조건 하에서 6 분 정도 학습하였지만 svmPoly 와 svmRadial 같은 경우엔 같은 학습 조건 하에서 약 40 분 가량 학습하였습니다.

이는 커널을 사용하는 방식의 계산량 증가가 변수 개수의 증가에 비례하기 때문이라고 생각합니다. svmPoly 의 경우, degree=2 일땐 변수 개수가 784 + 784H2 = 784 + 785C2 = 784 + 785*784/2 = 308,504 개이고 degree=3 일땐 변수 개수가 784 + 784H2 + 784H3 = 308,504 + 786H3 = 308,504 + 80,622,640 = 80,931,144 개로 degree 가 증가할 때 변수 개수가 기하급수적으로 증가하여 계산에 필요한 시간이 엄청나게 증가하는 것을 확인할 수 있습니다. 물론 원 문제를 푸는게 아닌 쌍대 문제를 커널을 이용하여 풀기 때문에 실제로 다항식에 대한 열을 추가시켜 svmLinear 를 적합시키는 방법보단 시간을 많이 줄인 것이지만 그에 비례해 계산량이 증가하는 것을 확인할 수 있었습니다. svmRadial 의 경우, 데이터 수만큼 열이 늘어나며 계산량이 증가하므로 svmLinear 보다 훨씬 많은 학습 시간이 소요되었습니다.

하이퍼 파라미터의 튜닝에 관해선, 적절한 범위에 대한 사전 정보가 없어서 임의로 범위를 설정하기보단 무작위 방법을 이용하였습니다. 무작위 방법을 위해 trainControl(search='random')을 설정해 주었습니다.

svmLinear 는 C = 0.1101521 일 때 0.9186653 를 validation accuracy 를 얻어 최적 하이퍼 파라미터를 얻었습니다. 그에 따른 test accuracy 는 0.926 입니다. test accuracy 가 더 높은 것으로 보아 train 에 과대 적합되진 않았지만 오히려 과소 적합을 의심해 볼 수 있습니다.

svmPoly 는 degree = 3, scale = 0.08564345, C = 0.5981654 일 때 0.9461656 를 validation accuracy 를 얻어 최적 하이퍼 파라미터를 얻었습니다. 그에 따른 test accuracy 는 0.951 입니다. 마찬가지로 test accuracy 가 더 높은 것으로 보아 train 에 과대 적합되진 않았지만 오히려 과소 적합을 의심해 볼 수 있습니다만 svmLinear 에 비해 성능이 개선되었고 절대적인 성능이 높다고 생각하여 문제가 없다고 판단하였습니다.

svmRadial 는 sigma = 0.02018301, C = 11.81963 일 때 0.9564990 를 validation accuracy 를 얻어 최적 하이퍼 파라미터를 얻었습니다. 그에 따른 test accuracy 는 0.958 입니다. Test accuracy 와 validation accuracy 의 차이가 0.002 이내인 것으로 보아 모델의 일반화가 잘 되었다고 생각합니다. 하지만 sigma 와 C 의 최적값이 모두 주어진 범위 내에서의 최댓값인 것으로 보아 탐색 범위를 넓혔을 때 더 좋은 하이퍼 파라미터를 얻을 가능성이 크다고 생각합니다. 전체 파라미터 공간 안에서 best 를 찾은 것은 아니지만 주어진 공간 내에서 optimal 한 값을 찾았다고 생각했고, svmLinear, svmPoly 보다 더 좋은 성능을 얻어서 하이퍼 파라미터 튜닝을 멈추고 최종 모델로 선택하였습니다.

Appendix : R code

```
##### load dataset

mnist_train = read.csv('./MNIST_train_small.csv', header=TRUE)
mnist_test = read.csv('./MNIST_test_small.csv', header=TRUE)

x_train = mnist_train[, 2:785]
y_train = as.factor(mnist_train[, 1])
x_test = mnist_test[, 2:785]
y_test = as.factor(mnist_test[, 1])

student = 20152410
##### construct support vector machine classifier

library(e1071)
library(caret)
library(kernlab)


method_list = c('svmLinear', 'svmPoly', 'svmRadial')
fold_number = 3
tune_length = 10
train_control = trainControl(method='cv',
                             number=fold_number,
                             search='random')
### linear svm

start = Sys.time()

set.seed(student)
linear_model = train(x_train,
                     y_train,
                     method=method_list[1],
                     trControl=train_control,
                     metric='Accuracy',
                     tuneLength=tune_length)
load_time = Sys.time() - start
load_time
## Time difference of 5.971888 mins
```

```
linear_model
```

```
## Support Vector Machines with Linear Kernel
##
## 6000 samples
##  784 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 3999, 4001, 4000
## Resampling results across tuning parameters:
##
##   C            Accuracy   Kappa
##     0.1101521  0.9186653  0.9095505
##     0.2812232  0.9116661  0.9017667
##     2.6407019  0.9050007  0.8943553
##     8.8305640  0.9050007  0.8943553
##    36.4625175  0.9050007  0.8943553
##    53.6442018  0.9050007  0.8943553
##    75.7921019  0.9050007  0.8943553
##   228.4851141  0.9050007  0.8943553
##   433.6071618  0.9050007  0.8943553
##   551.1111075  0.9050007  0.8943553
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.1101521.
```

```
linear_pred = predict(linear_model, newdata=x_test)
linear_table = table(linear_pred, y_test)
linear_cm = confusionMatrix(linear_table, mode='everything')
linear_cm
```

```
## Confusion Matrix and Statistics
##
##            y_test
## linear_pred   0   1   2   3   4   5   6   7   8   9
##           0  93   0   3   0   0   2   2   1   0   1
##           1   0 106   2   0   2   0   0   0   0   2
##           2   0   0 100   2   1   2   0   1   1   0
##           3   0   0   1  77   0   3   0   0   4   0
##           4   0   0   1   0 100   0   3   1   2   3
##           5   1   0   0   2   1  85   1   0   2   2
##           6   0   0   0   1   0   2  93   0   2   0
##           7   0   0   0   1   0   0   0 102   2   2
##           8   0   0   4   2   0   1   0   1  77   0
##           9   0   0   0   1   4   0   0   2   0  93
##
```

```
## Overall Statistics
##
##                Accuracy : 0.926
##                  95% CI : (0.908, 0.9415)
##     No Information Rate : 0.111
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9177
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9894   1.0000   0.9009   0.8953   0.9259   0.8947
## Specificity            0.9901   0.9933   0.9921   0.9912   0.9888   0.9901
## Pos Pred Value         0.9118   0.9464   0.9346   0.9059   0.9091   0.9043
## Neg Pred Value         0.9989   1.0000   0.9877   0.9902   0.9910   0.9890
## Precision              0.9118   0.9464   0.9346   0.9059   0.9091   0.9043
## Recall                 0.9894   1.0000   0.9009   0.8953   0.9259   0.8947
## F1                     0.9490   0.9725   0.9174   0.9006   0.9174   0.8995
## Prevalence             0.0940   0.1060   0.1110   0.0860   0.1080   0.0950
## Detection Rate         0.0930   0.1060   0.1000   0.0770   0.1000   0.0850
## Detection Prevalence   0.1020   0.1120   0.1070   0.0850   0.1100   0.0940
## Balanced Accuracy      0.9897   0.9966   0.9465   0.9433   0.9574   0.9424
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9394   0.9444   0.8556   0.9029
## Specificity            0.9945   0.9944   0.9912   0.9922
## Pos Pred Value         0.9490   0.9533   0.9059   0.9300
## Neg Pred Value         0.9933   0.9933   0.9858   0.9889
## Precision              0.9490   0.9533   0.9059   0.9300
## Recall                 0.9394   0.9444   0.8556   0.9029
## F1                     0.9442   0.9488   0.8800   0.9163
## Prevalence             0.0990   0.1080   0.0900   0.1030
## Detection Rate         0.0930   0.1020   0.0770   0.0930
## Detection Prevalence   0.0980   0.1070   0.0850   0.1000
## Balanced Accuracy      0.9669   0.9694   0.9234   0.9476
```

```r
### polynomial svm

start = Sys.time()

set.seed(student)
ploy_model = train(x_train,
                   y_train,
                   method=method_list[2],
                   trControl=train_control,
                   metric='Accuracy',
                   tuneLength=tune_length)

load_time = Sys.time() - start
load_time
```

```
## Time difference of 42.96775 mins
```

```r
ploy_model
```

```
## Support Vector Machines with Polynomial Kernel
##
## 6000 samples
##  784 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 3999, 4001, 4000
## Resampling results across tuning parameters:
##
##   degree  scale         C             Accuracy   Kappa
##   2       1.737694e-05    2.32211366  0.3176675  0.2304178
##   2       1.966788e-02    4.75246581  0.9389992  0.9321668
##   2       2.287321e-02    0.04064384  0.9178332  0.9086216
##   2       2.720802e-02  200.76500754  0.9408326  0.9342062
##   2       3.332947e-02    0.22875819  0.9426657  0.9362414
##   3       5.290869e-05    0.96647039  0.6203325  0.5759114
##   3       1.219221e-04    2.11694269  0.8718316  0.8573970
##   3       1.697705e-03    0.06655475  0.8289986  0.8096041
##   3       8.564345e-02    0.59816536  0.9461656  0.9401350
##   3       1.587170e-01  187.67195635  0.9456660  0.9395794
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 3, scale = 0.08564345 and C
##  = 0.5981654.
```

```r
ploy_pred = predict(ploy_model, newdata=x_test)
ploy_table = table(ploy_pred, y_test)
ploy_cm = confusionMatrix(ploy_table, mode='everything')
ploy_cm
```

```
## Confusion Matrix and Statistics
##
```

```
##          y_test
## ploy_pred   0   1   2   3   4   5   6   7   8   9
##         0  93   0   2   0   0   3   1   1   0   1
##         1   0 106   1   0   2   0   0   0   0   2
##         2   0   0 105   2   0   1   0   2   0   0
##         3   0   0   1  78   0   1   0   0   2   0
##         4   0   0   0   0 103   0   1   0   0   1
##         5   1   0   0   1   1  85   1   0   0   1
##         6   0   0   0   1   0   1  96   0   0   0
##         7   0   0   1   1   0   0   0 102   2   1
##         8   0   0   1   3   0   2   0   1  86   0
##         9   0   0   0   0   2   2   0   2   0  97
##
## Overall Statistics
##
##                Accuracy : 0.951
##                  95% CI : (0.9357, 0.9635)
##     No Information Rate : 0.111
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9455
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9894   1.0000   0.9459   0.9070   0.9537   0.8947
## Specificity            0.9912   0.9944   0.9944   0.9956   0.9978   0.9945
## Pos Pred Value         0.9208   0.9550   0.9545   0.9512   0.9810   0.9444
## Neg Pred Value         0.9989   1.0000   0.9933   0.9913   0.9944   0.9890
## Precision              0.9208   0.9550   0.9545   0.9512   0.9810   0.9444
## Recall                 0.9894   1.0000   0.9459   0.9070   0.9537   0.8947
## F1                     0.9538   0.9770   0.9502   0.9286   0.9671   0.9189
## Prevalence             0.0940   0.1060   0.1110   0.0860   0.1080   0.0950
## Detection Rate         0.0930   0.1060   0.1050   0.0780   0.1030   0.0850
## Detection Prevalence   0.1010   0.1110   0.1100   0.0820   0.1050   0.0900
## Balanced Accuracy      0.9903   0.9972   0.9702   0.9513   0.9757   0.9446
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9697   0.9444   0.9556   0.9417
## Specificity            0.9978   0.9944   0.9923   0.9933
## Pos Pred Value         0.9796   0.9533   0.9247   0.9417
## Neg Pred Value         0.9967   0.9933   0.9956   0.9933
## Precision              0.9796   0.9533   0.9247   0.9417
## Recall                 0.9697   0.9444   0.9556   0.9417
## F1                     0.9746   0.9488   0.9399   0.9417
## Prevalence             0.0990   0.1080   0.0900   0.1030
## Detection Rate         0.0960   0.1020   0.0860   0.0970
## Detection Prevalence   0.0980   0.1070   0.0930   0.1030
## Balanced Accuracy      0.9837   0.9694   0.9739   0.9675
```

```r
### radial svm

start = Sys.time()

set.seed(student)
radial_model = train(x_train,
                     y_train,
                     method=method_list[3],
                     trControl=train_control,
                     metric='Accuracy',
                     tuneLength=tune_length)

load_time = Sys.time() - start
load_time
```

```
## Time difference of 41.59553 mins
```

```r
radial_model
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 6000 samples
##  784 predictor
##   10 classes: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 3999, 4001, 4000
## Resampling results across tuning parameters:
##
##   sigma        C            Accuracy   Kappa
##   0.004451774   6.87490728  0.9396661  0.9329064
##   0.004779173   0.19853618  0.9011652  0.8900698
##   0.005353820   0.36343771  0.9148328  0.9052872
##   0.006283408   0.05255669  0.8518305  0.8351070
##   0.006547637   0.27951279  0.9148326  0.9052857
##   0.017157958   6.19099659  0.9551656  0.9501470
##   0.018133226  11.54186541  0.9558322  0.9508878
##   0.018391833   6.53863380  0.9559988  0.9510729
##   0.019130171   8.30370240  0.9563322  0.9514437
##   0.020183007  11.81963185  0.9564990  0.9516292
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.02018301 and C = 11.81963.
```

```r
radial_pred = predict(radial_model, newdata=x_test)
radial_table = table(radial_pred, y_test)
radial_cm = confusionMatrix(radial_table, mode='everything')
radial_cm
```

```
## Confusion Matrix and Statistics
##
```

```
##            y_test
## radial_pred   0   1   2   3   4   5   6   7   8   9
##           0  93   0   2   0   0   1   1   1   0   1
##           1   0 106   1   0   0   0   0   0   0   1
##           2   0   0 106   2   0   1   0   1   0   0
##           3   0   0   1  78   0   1   0   0   1   1
##           4   0   0   0   0 105   0   1   1   0   1
##           5   1   0   0   1   0  89   1   0   0   0
##           6   0   0   0   0   1   1  96   0   0   0
##           7   0   0   1   1   0   0   0 101   3   1
##           8   0   0   0   4   0   1   0   1  86   0
##           9   0   0   0   0   2   1   0   3   0  98
##
## Overall Statistics
##
##                Accuracy : 0.958
##                  95% CI : (0.9436, 0.9696)
##     No Information Rate : 0.111
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9533
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9894   1.0000   0.9550   0.9070   0.9722   0.9368
## Specificity            0.9934   0.9978   0.9955   0.9956   0.9966   0.9967
## Pos Pred Value         0.9394   0.9815   0.9636   0.9512   0.9722   0.9674
## Neg Pred Value         0.9989   1.0000   0.9944   0.9913   0.9966   0.9934
## Precision              0.9394   0.9815   0.9636   0.9512   0.9722   0.9674
## Recall                 0.9894   1.0000   0.9550   0.9070   0.9722   0.9368
## F1                     0.9637   0.9907   0.9593   0.9286   0.9722   0.9519
## Prevalence             0.0940   0.1060   0.1110   0.0860   0.1080   0.0950
## Detection Rate         0.0930   0.1060   0.1060   0.0780   0.1050   0.0890
## Detection Prevalence   0.0990   0.1080   0.1100   0.0820   0.1080   0.0920
## Balanced Accuracy      0.9914   0.9989   0.9752   0.9513   0.9844   0.9668
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9697   0.9352   0.9556   0.9515
## Specificity            0.9978   0.9933   0.9934   0.9933
## Pos Pred Value         0.9796   0.9439   0.9348   0.9423
## Neg Pred Value         0.9967   0.9922   0.9956   0.9944
## Precision              0.9796   0.9439   0.9348   0.9423
## Recall                 0.9697   0.9352   0.9556   0.9515
## F1                     0.9746   0.9395   0.9451   0.9469
## Prevalence             0.0990   0.1080   0.0900   0.1030
## Detection Rate         0.0960   0.1010   0.0860   0.0980
## Detection Prevalence   0.0980   0.1070   0.0920   0.1040
## Balanced Accuracy      0.9837   0.9642   0.9745   0.9724
```