Recognizing Handwritten Digits

SDS385
Elisa Ferracane - ef5884
Ashutosh Singh - as79592
Chakameh Jafari - cz5599

1 Summary

The goal of this project is to recognize handwritten digits 0-9 using the well-known MNIST dataset LeCun et al. [1998]. While very low test error rates have already been achieved (lowest is 0.23 using convolutional neural networks Ciregan et al. [2012]), this project uses only a subset of the data: 5000 for training, and 2500 for test. Because of the smaller size of the dataset, we speculated Support Vector Machine (SVM) might outperform a neural network, which is typically data-hungry. While we were able to achieve a test error rate of 0.0476 using a non-linear kernel with SVM (3rd degree polynomial), we did not have success implementing a neural network model for comparison.

2 SVM Models

We first removed predictors with little variability [1], resulting in dropping almost 41% (319) of the predictors. Next, a grid search with 5-fold cross validation was performed over the cost values [1, 5, 10, 1000] and gamma values [0.0001, 0.001, 0.01, 1]. We experimented with linear kernel, radial, sigmoid and polynomial of degrees 3-9, with the results recorded in Table 1. Strangely, a cost of 1 was often selected by the grid search, which typically yields poor results. We obtained the best results for cost=1000, gamma=0.01 and polynomial kernel of degree 3. Finally, we created a model with these tuned parameters and tested on the test set. The resulting test error rate was **0.0476**. However, we did not conduct a statistical test to determine whether this result is statistically significantly better than the next-best model using the radial kernel with 0.0484. The R code is included in the Appendix.

Kernel	Gamma	Cost	Train error rate	Test error rate
polynomial deg=3	0.01	1	0.0592	0.22
polynomial deg=5	0.01	1	0.1284	0.22
polynomial deg=7	0.01	1	0.2612	0.22
polynomial deg=9	0.01	1	0.406	0.22
polynomial deg=9	0.01	1000	0.0498	**0.0476**
radial	0.001	5	0.0596	0.0484
sigmoid	0.001	1	0.091	0.0636

Table 1: Error rates for different SVM models, with parameters determined by grid search.

3 Other models

We further attempted Lasso and a feed-forward network, but encountered errors programming them in R. Finally, we took a different approach to eliminating predictors with little variability by first doing a principal components analysis and considering only the first 75 components. These were then used as input to an SVM model with a radial kernel. Although we achieved a superior test error rate of 0.02, this code was implemented in Python using the full dataset, and we were not able to replicate the results in R.

References

Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[1] SD<20, this parameter was set by tuning on the training set.

Appendix

```
#read part of data
train_rows = 5000
test_rows  = 2500

train = read.csv("train.csv", header = F, nrows=train_rows )
test  = read.csv("test.csv", header = F, nrow=test_rows )

#convert last column to factor
train$V785 = as.factor(train$V785)
test$V785  = as.factor(test$V785)

#remove predictors with little variability
###
### 20 was sort of a guess!
###
cut_off     = 20
bad_columns = which(apply(train[,-ncol(train)], 2, sd) < cut_off)
length(bad_columns)

train = train[,-bad_columns]
test  = test[,-bad_columns]

#train and test with SVM
library(e1071)

#tune
set.seed(1010)
system.time({
  tune_svm = tune(svm , V785 ~ ., data = train, kernel ="polynomial", degree=3,
                  ranges =list(cost = c(1 ,5,10,1000), gamma = c( 0.0001, 0.001, 0.01, 1 )),
                  tunecontrol = tune.control(cross = 5) )
})

tune_svm

#create final model with tuned parameters
svm_final   = svm(V785 ~ ., data = train, cost = tune_svm$best.parameters$cost, kernel="polyno
                  gamma = tune_svm$best.parameters$gamma)
svm_y_final = predict(svm_final, newdata = test)

confusion_matrix_final = table(predicted = svm_y_final, actual = test$V785)
confusion_matrix_final

error_rate_final = 1 - ( sum(diag(confusion_matrix_final)) / nrow(test))
error_rate_final
```