STATS 503 HW #5 for Group 2

Katherine Ahn, Haonan Feng, Diana Liang, and Karen Wang

Due on: 4/6/2020

1. **Solution to Optimization**

$$\min_{\beta_0, \beta} \sum_1^n [1 - y_i f(x_i)]_+ + \frac{\lambda}{2} \|\beta\|^2$$

$$\Big\downarrow \ \lambda = \frac{1}{C}$$

$$\min_{\beta_0, \beta} \sum_1^n [1 - y_i f(x_i)]_+ + \frac{1}{2C} \|\beta\|^2$$

$$\Big\downarrow$$

$$\min_{\beta_0, \beta} C \underbrace{\sum_1^n [1 - y_i f(x_i)]_+}_{\xi_i} + \frac{1}{2} \|\beta\|^2$$

Let $\xi_i = [1 - y_i f(x_i)]_+$

① $[1 - y_i f(x_i)]_+ \geq 0$

② $[1 - y_i f(x_i)]_+ \geq 1 - y_i f(x_i)$

$$\Rightarrow \begin{cases} \xi_i \geq 0 \\ \xi_i \geq 1 - y_i(\beta_0 + x_i^T \beta) \end{cases}$$

$$\therefore \min_{\beta_0, \beta} \sum_1^n [1 - y_i f(x_i)]_+ + \frac{\lambda}{2} \|\beta\|^2$$

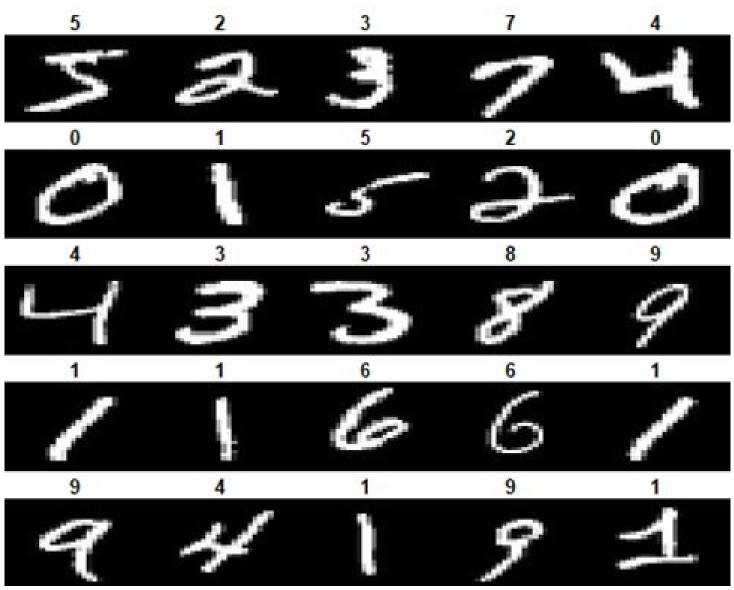
$$\equiv$$

$$\min_{\beta_0, \beta} \frac{1}{2} \|\beta\|^2 + C \sum \xi_i \quad \text{subject to } \xi_i \geq 0, \ y_i(\beta_0 + x_i^T \beta) \geq 1 - \xi_i, \ \forall i$$

2. **Digits in MNIST Dataset**

We first did some exploratory data analysis to get an idea of what our data looks like and its features.

```
mnist <- load("mnist.Rdata")
# Swap the train and test data
temp = x_test
x_test = x_train
x_train = temp
temp = y_test
y_test = y_train
y_train = temp
# Visualize the first 25 samples
class_names = seq(0,9,by=1)
par(mfcol=c(5,5))
par(mar=c(0, 0, 1.5, 0), xaxs='i', yaxs='i')
for (i in 1:25) {
  img <- x_train[i, , ]
  img <- t(apply(img, 2, rev))
  image(1:28, 1:28, img, col = gray((0:255)/255), xaxt = 'n', yaxt = 'n',
        main = paste(class_names[y_train[i] + 1]))
}
```



a. Support Vector Machine Classifier

First we standardize and flatten the data.

```
x_train <- x_train / 255
x_test <- x_test / 255
x_train_flat <- matrix(x_train, dim(x_train)[1], prod(dim(x_train)[2:3]))
x_test_flat <- matrix(x_test, dim(x_test)[1], prod(dim(x_test)[2:3]))
y_train = as.factor(y_train)
y_test = as.factor(y_test)
train_dat = data.frame(y_train, x_train_flat)
colnames(train_dat)[1] = "labels"
test_dat = data.frame(y_test, x_test_flat)
colnames(test_dat)[1] = "labels"
```

Apply cross validation and tune the parameters of SVM.

```
set.seed(1)
tune.out =tune.svm(labels~., data=train_dat, cost=c(0.1,1), degree=c(1,2),
kernel=c("radial", “polynomial”), gamma=c(0.001,0.1), cross=5)
summary(tune.out)
```

Model(Radial/Polynomial)	Argument(cost, degree, gamma in order)	CV Error
radial	0.1, 1, 0.001	0.1476
radial	0.1, 1, 0.1	0.7359
radial	1.0, 1, 0.001	0.0810
radial	1.0, 1, 0.1	0.1134
Polynomial	1.0, 2, 0.1	0.0344
Polynomial	1.0, 2, 0.001	0.3069
Polynomial	0.1, 2, 0.1	0.0350
Polynomial	0.1, 2, 0.001	0.8864
Polynomial	0.1, 1 , 0.001	0.233
Polynomial	0.1, 1 , 0.1	0.0675

Polynomial	1, 1 , 0.001	0.0921
Polynomial	1, 1 ,0.1	0.0654

So we choose cost = 1.0, degree = 2, gamma = 0.1, kernel = polynomial.

```
test_pred = predict(tune.out$best.model, newdata = test_dat)
# confusion matrix using the optimal SVM
table(test_pred, test_dat$labels)


 test_pred    0    1    2    3    4    5    6    7    8    9
        0 5786    1   18    8   14   12   20   10   19   32
        1    1 6624   27   29   19   25   14   23   84   10
        2   26   37 5685  111   37    9   28   32   73   19
        3    8    9   31 5698    1   61    1    4   58   47
        4    5   12   37    6 5625   13   18   35   23  105
        5   24    6    7  110    0 5168   56    1   69   18
        6   25    3   27   12   30   64 5765    5   29    1
        7    7   23   76   48   20    4    0 6070   20  120
        8   28   13   39   67    6   38   15    9 5420   23
        9   13   14   11   42   90   27    1   76   56 5574

# test error of the optimal SVM
mean(test_pred != test_dat$labels)
## [1] 0.04308333
```

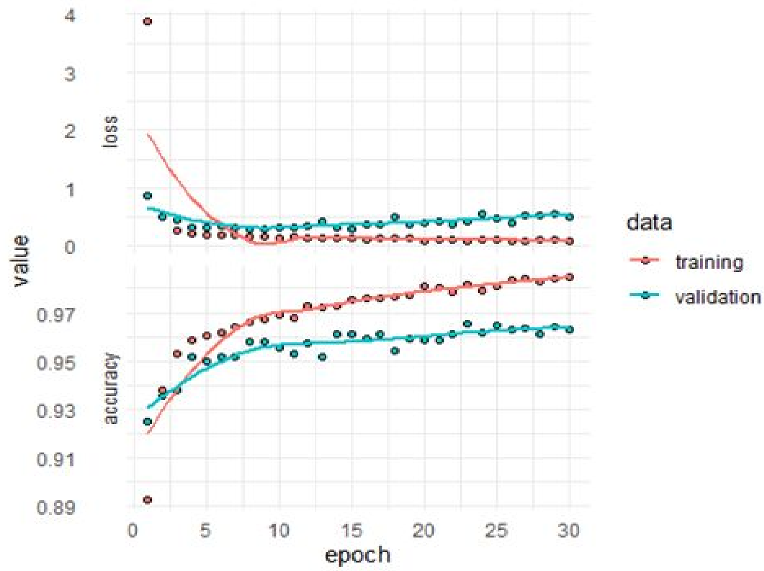
b. *MLP and CNN*

For the MLP model, we tried these hyperparameters: number of hidden units(128, 256, 384), number of epochs(10, 30, 50), and batch size(16, 32, 64). The best model is below.

```
mlp_model <- keras_model_sequential()
mlp_model %>%
  layer_flatten(input_shape = c(28, 28)) %>%
  layer_dense(units = 286, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax') %>% compile(
  optimizer = 'adam',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
)
mlp_m = mlp_model %>% fit(x_train, y_train, epochs = 30,
validation_split = 0.2, batch_size = 64)
```

```
mlp_score <- mlp_model %>% evaluate(x_test, y_test)
```

This is the plot of training and validation loss and accuracy in fitting the MLP model.



For the CNN model, we tried the following hyperparameters: kernel size(3, 4), activation function(relu, sigmoid, tanh), and dropout(0.25, 0.40). The best model is below:

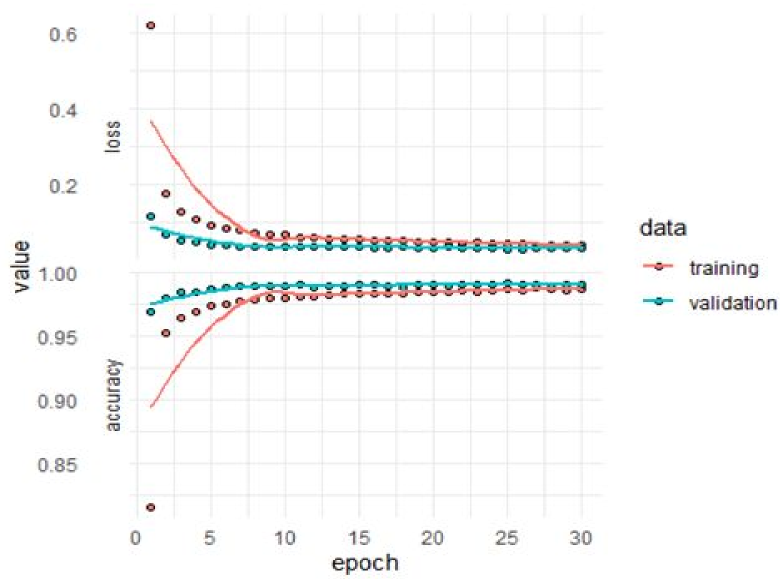
```
cnn_model <- keras_model_sequential()
cnn_model %>%
layer_conv_2d(filter=32,kernel_size=c(4,4),padding="same",input_s
hape=c(28,28, 1) ) %>%  layer_activation("sigmoid") %>%
layer_max_pooling_2d(pool_size=c(2,2)) %>%
layer_conv_2d(filter=32,kernel_size=c(4,4))  %>%
layer_activation("sigmoid") %>%
layer_max_pooling_2d(pool_size=c(2,2)) %>%
layer_dropout(0.25) %>%
layer_flatten() %>%
layer_dense(64) %>%
layer_activation("sigmoid") %>%
layer_dropout(0.5) %>%
layer_dense(10) %>%
layer_activation("softmax")

cnn_xtrain = array(x_train, dim = c(dim(x_train)[1],
dim(x_train)[2], dim(x_train)[3], 1))
```

```
cnn_xtest = array(x_test, dim = c(dim(x_test)[1], dim(x_test)[2],
dim(x_test)[3], 1))

cnn_model %>% compile(
  optimizer = 'adam',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
)
cnn_m = cnn_model %>% fit(cnn_xtrain, y_train, epochs = 30,
validation_split = 0.2, batch_size = 32)
cnn_score <- cnn_model %>% evaluate(cnn_xtest, y_test)
```

The is the plot of training and validation loss and accuracy in fitting the CNN model.



This is our table of validation and test accuracy for our best models.

Model Type	Validation Accuracy	Test Accuracy
MLP	0.9628333	0.9666
CNN	0.9905834	0.9909

Our MLP model had a similar validation/CV error of around 0.04 and similar test errors of around 0.04. The **CNN model** performed better than the SVM and the

MLP with a validation error and test error of less than 0.01. The SVM model, however, was only fitted using what was originally labelled x_test (we have swapped the train and test data in part a, but not part b) and, thus, did not have as many observations to fit to. Since the MLP and CNN models were allowed more observations to fit to, this comparison may not truly reflect the abilities of these modeling techniques.